TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

CHAO HAN

OPENSTACK BASED PACKET CONTROL UNIT CLOUDIFICATION

Master of Science thesis

Examiner: Prof. Mikko Valkama
Supervisor: Tuomo Eskola
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 5th November 2014

ABSTRACT

CHAO HAN: OpenStack Based Packet Control Unit Cloudification
Tampere University of Technology
Master of Science thesis, 54 pages, 16 Appendix pages
Tampere 2015
Master's Degree Programme in Information Technology
Major: Communication Systems and Networks
Examiner: Prof. Mikko Valkama
Keywords: cloud computing, cloud platform, cloud networks, cloud deployment, OpenStack, Packet Control Unit

With the increasing number of mobile devices in use, the huge amount of data keeps growing and brings tough challenges to the telecommunication operators. Operators expect to guarantee and improve their service quality at lower Capital Expenditures (CAPEX) and Operable Expenditures (OPEX). However, as the existing software is designed for the dedicated hardware infrastructure, it complicates the developing procedure of the software. Moreover, it also takes a long period to deliver the new version of products. In addition, the hardware resources cannot be allocated on-demand, which causes the waste of the resources.

As the cloud computing could be the possible solution to solve the problems, the thesis introduces a cloud platform powered by OpenStack to facilitate the development of cloudified products. The OpenStack based cloud platform is deployed on the top of the physical hardware running the Linux operating system. According to the running condition of the Packet Control Unit (PCU), the target product to be cloudified, a testing environment is established based on the cloud platform. The PCU and its relevant components are provisioned and tested in the cloud.

The cloudified PCU is demonstrated to be successful that it can run stably in the OpenStack based cloud environment. The whole cloud environment is virtualized, which means the software can be decoupled with the physical hardware. The new version of the software can be delivered fast and simply in the cloud. The centralized cloud resource pool can be elastically allocated on-demand. If more services provided by OpenStack can be utilized and a reconstruction of PCU software can be done in future works, a fully cloudifed product will be accomplished.

PREFACE

I feel privileged to acquire this thesis topic from Nokia Networks. With the instructing by Tuomo Eskola, my supervisor, and strong support from the team when doing the thesis work at Nokia Networks, I could complete my thesis work successfully at last. Thereby, I would express my great gratitude to the whole team, especially to Tuomo. The experience at Nokia Networks has become one of my most memorable time.

I am grateful to Professor Mikko Valkama, who examined my thesis and provided helpful advice for the thesis schedule. I would say, without his elaborate guidance, the thesis cannot be finished in time.

At last, I would appreciate to all my families and friends who have ever encouraged me to make this accomplishment.

Tampere, 15.7.2015

Chao Han

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

API	Application Programming Interface
BSC	Base Station Controller
BSSGP	Base Station Subsystem GPRS Protocol
BTS	Base Transceiver Station
CAPEX	Capital Expenditures
DHCP	Dynamic Host Configuration Protocol
ETSI	European Telecommunications Standards Institute
FWaaS	Firewall as a Service
GB	Gigabytes
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GRE	Generic Routing Encapsulation
GSM RF	GSM Radio Frequency
GUI	Graphical User Interface
GSM	Global System for Mobile Communication
HLR	Home Location Register
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IP	Internet Protocol
IT	Information Technology
KVM	Kernel based Virtual Machine
LAN	Local Area Network
LBaaS	Load-Balancer as a Service
LLC	Logical Link Control
MAC	Media Access Control
MB	Megabytes
ML2	Module Layer 2
MS	Mobile Station
NIST	National Institute of Standards and Technology
NS	Network Services
OPEX	Operable Expenditures
OS	Operating System
OVS	Open Virtual Switch
PaaS	Platform as a Service
PCU	Packet Control Unit
PCU-CI	PCU Continuous Integration

PSTN	Public Switched Telephone Network
QEMU	Quick Emulator
RAN	Radio Access Network
RHEL	Red Hat Enterprise Linux
RLC	Radio Link Control
SaaS	Software as a Service
SGSN	Serving GPRS Support Node
SLA	Service-Level Agreement (between service providers and service users)
SNDCP	Sub Network Dependent Convergence Protocol
SSH	Secure Shell
UE	User Equipment
UI	User Interface
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VLR	Visitor Location Register
VXLAN	Virtual Extensible Local Area Network
VM	Virtual Machine

1. INTRODUCTION

In the beginning, the motivation of launching the thesis work is explained. The objectives of the project as well as how it is accomplished are generally introduced afterwards. The outline of the thesis is highlighted at last.

1.1 Motivation

Thanks to the development of the mobile communication technology, more and more mobile devices are used in people's daily life. A substantial market share goes to the telecommunication operators. Meanwhile, due to the growing quantity of the mobile data, operators also have to face some tough challenges. As the current software is designed for the dedicated physical hardware, a reconstruction of the software is mandatory to meet the standard of the new generation of physical hardware. This is not a good cycle of the development. Operators would like to become independent of the hardware infrastructure. In addition, to deliver a new product, operators have to order the physical hardware and provision the software from the scratch. As the procedure of completing these will take over months, it critically affects the release cycle. Therefore, operators expect to shorten the time of delivering new products. Moreover, the physical resource will be all reserved once the software is provisioned. This means the idling resource of the physical hardware cannot be allocated flexibly. However, operators wish to acquire the ability of dynamic scaling for the resources. All in all, operators expect to guarantee and improve their service quality at lower Capital Expenditures (CAPEX) and Operable Expenditures (OPEX).

To address the concerns from operators, telecommunication vendors turn their attention to cloud computing. With the cloud computing concept, hardware can be decoupled with operators. In such case, operators will only purchase their services on a software level. Also, cloud computing is a so-called "pay-as-you-go" model, operators will only buy the services on-demand. Furthermore, the time of deploying products will be significantly reduced in the cloud environment. In a word, introducing the cloud computing technology to the telecommunication world is a certain trend in changing the current situation.

1.2 Objectives

As the concerns have been depicted in the previous section, the target of this thesis is to build up a cloud testing environment from a vendor's point of view. As the core unit of General Packet Radio Service (GPRS) system in Global System for Mobile Communications (GSM) radio network, Packet Control Unit (PCU) will be the main object that is going to be cloudified. To reach the target, OpenStack[1] will be the main method for deploying the cloud platform. Besides, Linux based Operating System (OS) is used as the host OS, where OpenStack is built on top of that. Kernel-based Virtual Machine (KVM)[2] with Quick Emulator (QEMU)[3] as the hypervisor provides the prerequisite for the cloud setup. Figure 1.1 shows the cloud hierarchy for PCU.

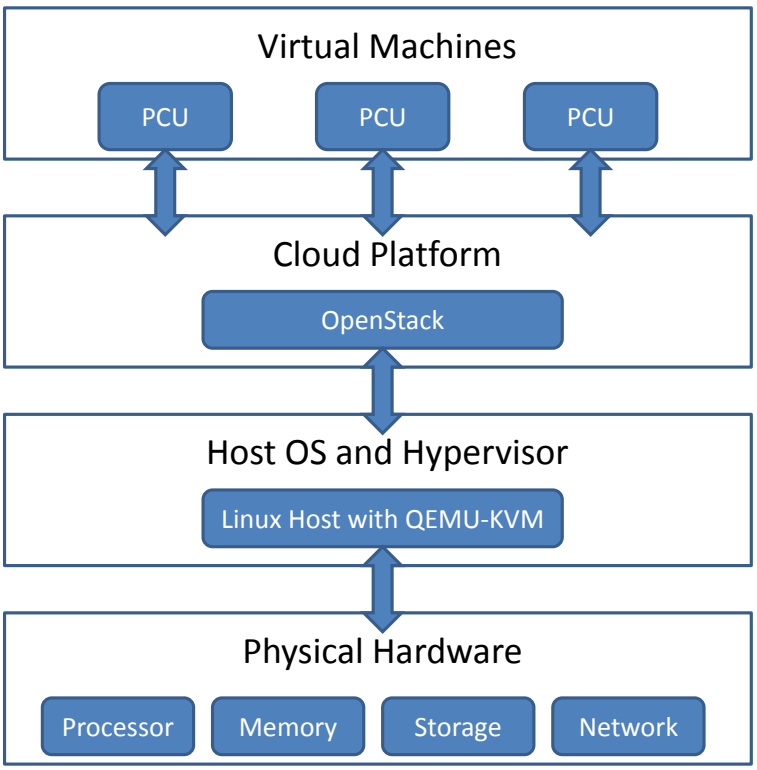

Figure 1.1 *Cloud hierarchy for PCU*

[1] http://www.openstack.org/
[2] http://www.linux-kvm.org/page/Main_Page
[3] http://wiki.qemu.org/Main_Page

1.3 Contents and structures

The thesis starts with the basic background introduction of cloud computing in Chapter 2, as it is the main topic that is going to be researched in the thesis. Different cloud infrastructure models, virtualization and cloudification are depicted. In Chapter 3, the main utilized object, PCU, is explained. GPRS networks as a background of PCU is described in general as well. Chapter 4 gives an introduction for OpenStack and its services that can provide. The way of deploying a plain OpenStack based cloud platform is also explained as follows. In Chapter 5, the establishment process of the cloud environment for PCU is stated in detail. At last, Chapter 6 gives a summary of the research work for the thesis.

2. CLOUD COMPUTING

As the main concept that this thesis will utilize, this chapter will focus on the topic of cloud computing. The general idea of cloud computing will be depicted at first. The knowledge about virtualization and cloudification will be presented as follows to give a deeper view.

2.1 Overview of cloud computing

Although cloud computing is a quite popular technical terminology nowadays, it can be dated back to fifties when the first mainframe computer was born. At that time, devices were directly connected to the computer using proprietary communication protocols. Later, minicomputers were developed, allowing access to computing resources from different locations through Local Area Network (LAN). Then servers rapidly evolved and gradually formed a data center, which is the central brain that can be accessed from anywhere whenever you need it [10]. This is the brief history that how cloud computing was evolved.

One modern accepted concept of cloud computing was defined by National Institute of Standards and Technology (NIST)[1], where it mentions: "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [13]." Some essential characteristics can be summarized as follows to be the benefit of cloud computing:

- On-demand self-service: Computing capabilities can be provisioned by consumers themselves without any interaction with the service provider.
- Broad network access: Capabilities can be acquired through the standard network by using any client platforms.

[1] http://www.nist.gov/

- Resource pooling: By using a multi-tenant model, consumers will be served in a shared computing resource pool, where physical and virtual resources are dynamically assigned according to the demand of consumers.
- Rapid elasticity: Capabilities can be provisioned elastically in any quantity at any time without constraints.
- Measured service: Resource usage can be metered for both service providers and service consumers.

2.1.1 Service models

According to different types of services that cloud computing is able to provide, it can be categorized into different classes. A service model with three abstract layers is the most commonly known model as depicted in Figure 2.1. From bottom to top, they are "Infrastructure as a Service" (IaaS), "Platform as a Service" (PaaS) and "Software as a Service" (SaaS). However, it does not mean these three are the only existing services. Nowadays, "XaaS" is a common way to describe a service model in Information Technology (IT) business field, such as "Firewall as a Service" (FWaaS), "Load-Balancer as a Service" (LBaaS) and even "Everything as a Service". However, we will only go through the basic well-known model in the thesis.

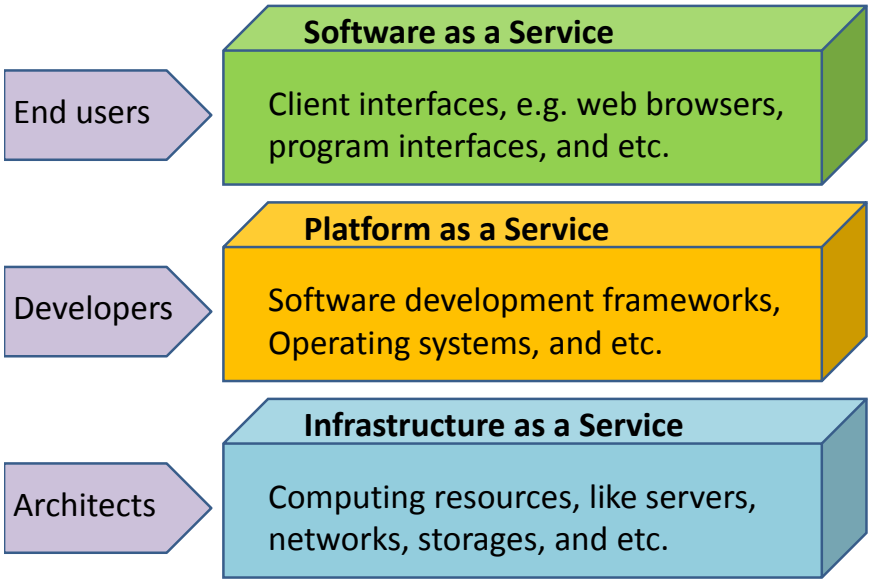

Figure 2.1 *Cloud computing service model*

IaaS

IaaS is the bottom layer where infrastructure architects can deploy or run arbitrary software on the computing resources, like servers, storages, networks or any other fundamental resources, which are supplied by service providers. End users have control over the OSs, storages, and deployed software, but do not have the control of the underlying cloud infrastructure [13]. The physical infrastructure resources are virtual resources which are hidden from the end users. Thus, end users do not need to maintain the physical servers at all.

Some benefits that can be acquired from IaaS [2]:

- Web access to the resources: Required infrastructure can be easily accessed via web browsers.
- Centralized management: Distributed resources are managed in a single management console.
- Elasticity and dynamic scaling: Depending on the load, infrastructure resources can be scaled dynamically.
- High resources utilization: Multiple users will share the same physical infrastructure.
- Preconfigured virtual machines (VMs): VMs are ready to be used directly.
- Metered services: The usage of the infrastructure resources are measurable, users will be charged based on that. Basically, it goes in a "pay-as-you-go" business model.

Some well-known IaaS providers are Amazon Web Services[2], Eucalyptus[3], Apache CloudStack[4], OpenStack, etc. In the thesis, OpenStack is used to provide an IaaS solution, it will be introduced more in the later chapter.

PaaS

PaaS is the middle layer based on the cloud infrastructure. It is used by the developers to deploy and develop applications on the development platform provided

[2]http://aws.amazon.com/
[3]https://www.eucalyptus.com/
[4]http://cloudstack.apache.org/

by the service providers. Developers are exempted from managing the development platforms and the underlying infrastructure, but they have the control of the deployed applications and the configuration of the development environment. Service providers in this level may provide programming languages, application frameworks, and developing tools over the Internet. In such a case, developers could develop applications online and deploy them immediately on the same platform. This can lower the time required for maintenance among different development tools [2].

Some of characteristics based on PaaS:

- All in one: Service providers offer all developing and maintaining services in one Integrated Development Environment (IDE).
- Web access to the development platform: By using web based User Interface (UI), development platforms can be accessed easily.
- Offline access: Local IDE could be synchronized with the PaaS service.
- Build-in scalability: Applications developed based on PaaS are capable of handling varying loads.
- Collaborative platform: Common platform is available for developers that are working on the same project.
- Diverse client tools: Wide variety of client tools can be chosen by developers.

Some popular providers in PaaS are Google App Engine[5], Cloud Foundry[6], Microsoft Windows Azure[7], etc.

SaaS

On the top layer, SaaS is used to run cloud applications where it interacts with the end users. Applications are accessible via different kinds of client devices, either a thin client interface, e.g. a web browser, or a program interface. End users do not control the underlying cloud infrastructure and are limited to making configurations for applications [13]. SaaS delivers on-demand services over the Internet, end users can access to the application according to their needs.

Some features for SaaS are as follows [2]:

[5]https://cloud.google.com/appengine/docs/whatisgoogleappengine?csw=1
[6]http://www.cloudfoundry.org/index.html
[7]http://azure.microsoft.com/en-us/

- One to many: Multiple tenants can share a single instance of the application.
- Web access: End users can access their applications whenever their device is connected to the Internet.
- Centralized management: SaaS service providers control the software centrally. It ensures every end user to be served with the same version of the application.
- Multi-device support: Any client-side devices are able to access services.
- Better scalability: SaaS applications work efficiently by dynamically scaling the underlying cloud resource.
- High availability: Proper backup and recovery mechanisms are implemented at the back-end to safeguard against the loss of the user data.
- Application Programming Interface (API): Through the standard APIs, other software and services can be integrated.

Some SaaS providers that people commonly know: Google Apps[8], Microsoft Office 365[9], etc.

2.1.2 Deployment models

With different ways of setting up the cloud environment, cloud computing also can be divided into various deployment models. This is a very important concept, because the deployment model should be decided based on the requirement. Otherwise, an unsuitable deployment model may affect the organization quite heavily. Generally, there are four basic types of deployment models: Private cloud, Public cloud, Community cloud and Hybrid cloud.

Private cloud

According to the definition of NIST, a private cloud infrastructure is deployed for exclusive use by a single organization, where the single organization is comprised of multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may also exist on or off premises [13]. The size of the private cloud is small compared to the

[8]https://www.google.com/work/apps/business/
[9]https://login.microsoftonline.com/, personal account is needed for sign-in.

other cloud models. Only users belonging to the organization have access to the cloud service, others from outside of the organization cannot access it. In the thesis, a private cloud is built for the current PCU testing environment. Figure 2.2 well depicts a private cloud model.

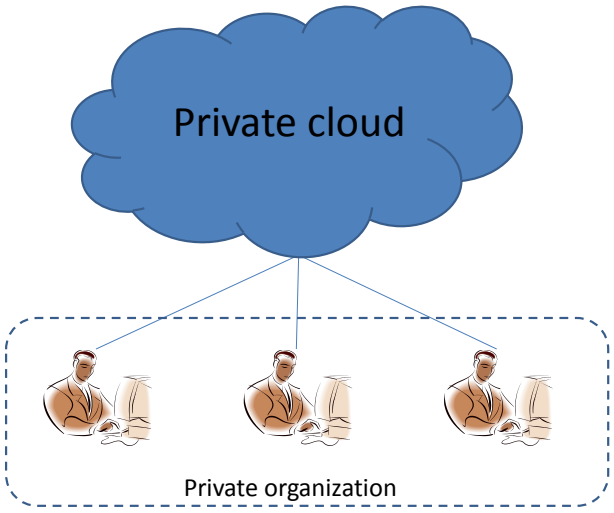

Figure 2.2 *Private cloud model*

Some features of the private cloud are listed as follows [2]:

- Secure: As a private cloud is deployed and managed by the organization itself, the possibility of leaking data out of the cloud is low.
- Central control: The organization has full control over the private cloud, it does not rely on anybody else.
- Weak service-level agreements (SLAs): Formal SLAs may or may not exist in a private cloud. Even if they exist, they will be weak, because the organization and users are in the same organization.

Public cloud

Still based on the definition of NIST, a public cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or a combination of them. It exists on the premises of the cloud provider [13]. In a word, a public cloud is open

to users from all over the world, and users do not have to be restricted in a single organization. A public cloud can be described with Figure 2.3.

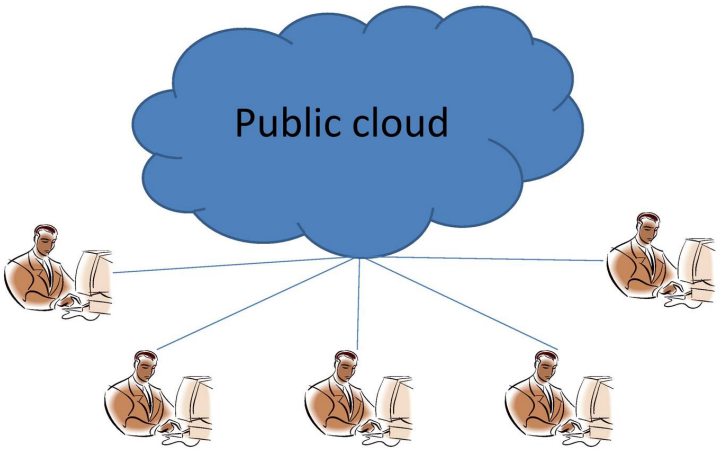

Figure 2.3 *Public cloud model*

Characteristics of the public cloud are stated below [2]:

- Highly scalable: Resources in a public cloud are considered as infinite, and service providers guarantee the grant for all requests.
- Affordable: Public cloud is charged as the way of "pay-as-you-go" model, users will be charged only for the service that they used.
- Less secure: Public cloud is offered by a third party, which has full control of the cloud. Thus, there will be a higher risk of data being leaked.
- Highly available: With proper permission, anyone can access the public cloud anywhere and anytime.
- Stringent SLAs: SLAs are strict for service providers and customers.

Community cloud

By the definition of NIST, a community cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations who share concerns (e.g., missions, security requirements, policies, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off

premises [13]. This could be considered as an extension of the private cloud, but shared by different organizations. Figure 2.4 is illustrated as a community cloud model.

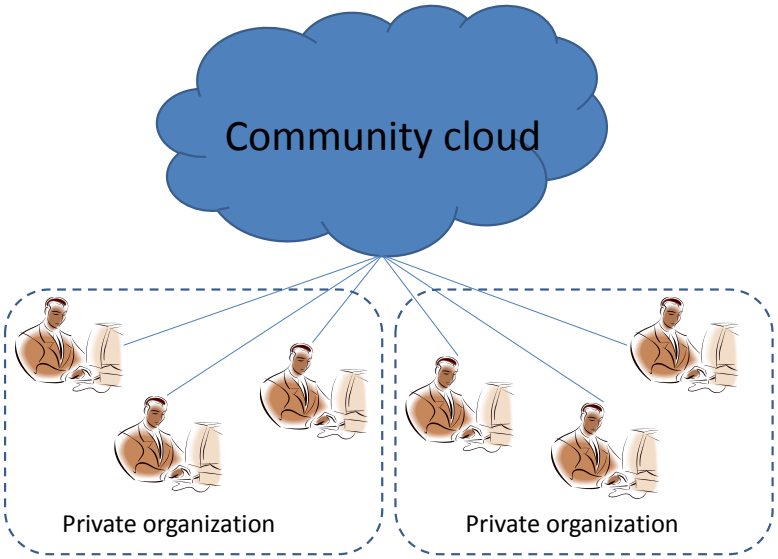

Figure 2.4 *Community cloud model*

A community cloud has the following features [2]:

- Collaborative and distributive maintenance: No single party has full control of the entire cloud.
- Partially secure: As the cloud is shared by several organizations, it is insulated from the outside world. However, the data may be leaked to other organizations within the community.
- Cost effective: Responsibilities and costs are shared by organizations in the same community.

Hybrid cloud

Hybrid cloud is defined by NIST as the cloud infrastructure that is a composition of two or more distinct cloud infrastructures (private, public, or community) which remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds) [13]. A hybrid cloud combines the advantages of both private cloud and public cloud. When there is the real product in the future, a

hybrid cloud could be set up for the operators. Figure 2.5 is depicted for a hybrid cloud model.

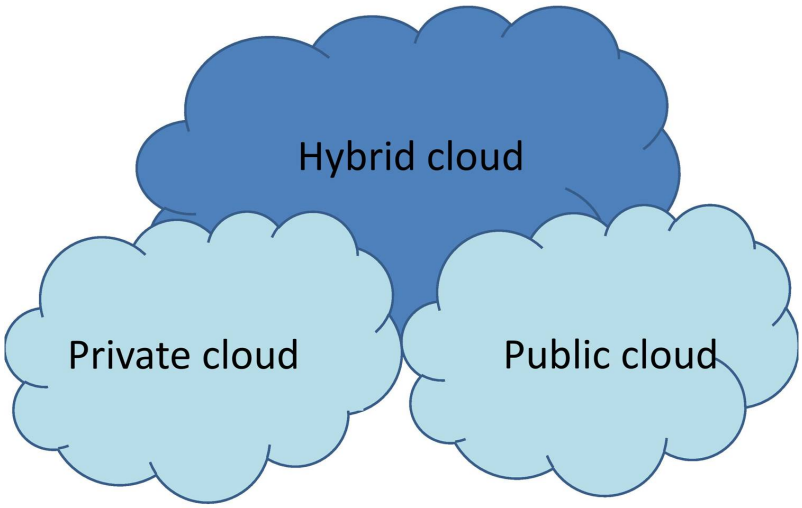

Figure 2.5 *Hybrid cloud model*

Characteristics of the hybrid cloud are as follows [2]:

- Scalable: As the hybrid cloud is combined with public cloud and private cloud, it inherits the scalability of public cloud.
- Partially secure: Same as above, hybrid cloud is partially secure due to the presence of a public cloud.
- Stringent SLAs: More stringent SLAs than private cloud.
- Complex cloud management: As a hybrid cloud contains more than one type of deployment models and there might be more users, the cloud management will be more difficult.

2.2 Virtualization

Virtualization is a technology with which the single physical hardware resources, such as processors, memories, networks, storages, and etc., can be used to run with different OSs and applications. It is the enabler for the cloud computing. Without virtualization, there is no need to talk about the cloud computing at all. This technology was firstly introduced by IBM, who defined the virtual machine concept in the 1960s. At that moment, it enabled time sharing and resource sharing on the very

expensive hardware. Afterwards, throughout the 1990s, virtualization was recalled back to the world due to the needs to run different OSs or applications on a single machine. With virtualization, both hardware and software can be virtualized. Thus, end users are able to use all resources. This significantly improves the efficiency of resource utilization [31].

2.2.1 Hypervisors

To make the hardware virtualization, hypervisor is the core element to be mentioned. It is the middleware between the physical hardware and the virtual machine or guest OS, which can provide the required resources for VMs. It allows the opportunity to map one physical hardware to a number of different VMs. There are many different hypervisors which are used in the virtualization world, but they mainly can be classified into two types: the "bare metal hypervisor", and the "hosted hypervisor".

Bare metal hypervisor

Bare metal hypervisor is also recognized as type 1 hypervisor. It can run and access the physical hardware directly without help from the host OS, like Figure 2.6 depicts. In this type, the additional overhead caused by the communication with host OS is saved. Therefore, it has better efficiency than the hosted hypervisor. This type of hypervisor can be used for servers that handle heavier loads and require higher security [2].

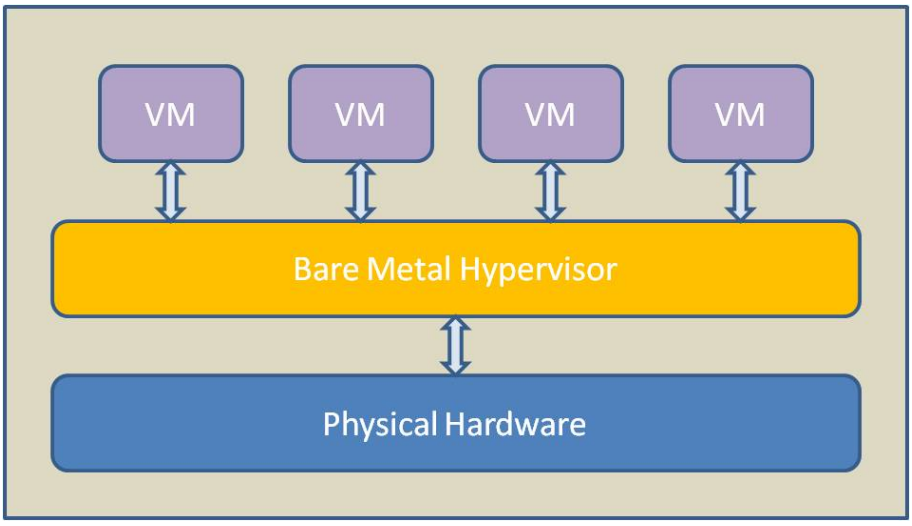

Figure 2.6 *Bare metal hypervisor*

VMware ESX is a good example for the bare metal hypervisor. It can abstract resources into multiple VMs which run unmodified OSs and applications [35].

Hosted hypervisor

Hosted hypervisor is known as type 2 hypervisor. It is not like the type 1 hypervisor, which allows to run and directly access the physical hardware. Instead, the host OS is required between the physical hardware and the hypervisor, as shown in Figure 2.7. The hypervisor is installed actually as a software on top of the host OS. This type of hypervisor is easily managed compared to the type 1 hypervisor, but with lower efficiency as the host OS is located for communication between the hypervisor and the physical hardware.

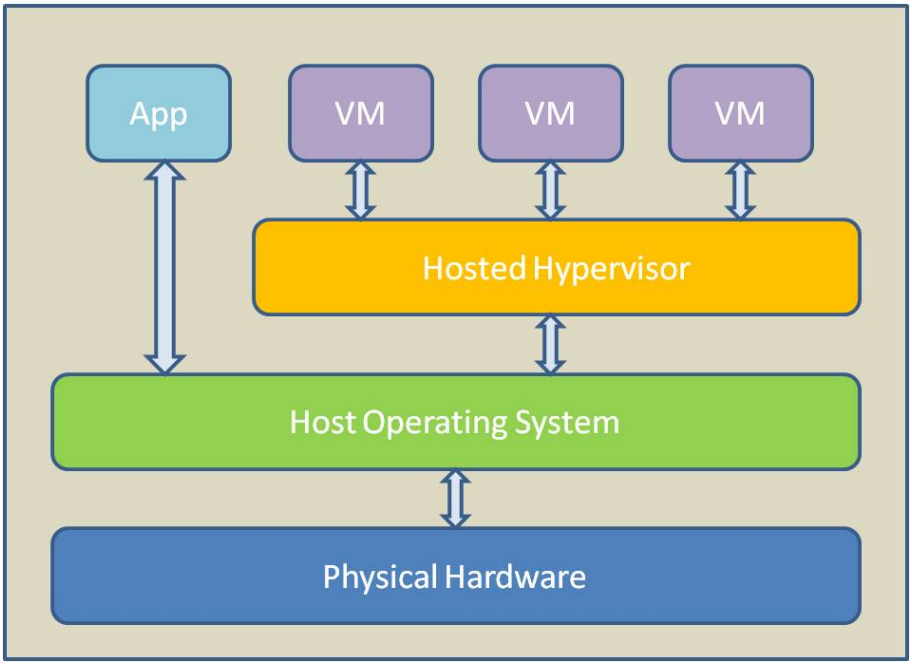

Figure 2.7 *Hosted hypervisor*

KVM is the typical type 2 hypervisor for Linux. It is merged into the Linux kernel machine. In this thesis, KVM is the type of hypervisor that is used for virtualization.

2.2.2 Virtualization techniques

When it comes to the virtualization techniques, the privileged rings are the very important thing to get familiar with at first. Privileged rings are separated by levels from 0 to 4, where ring 0 has the highest privilege that can access the physical

hardware directly; in contrast, ring 3 has the least privilege and normally runs the user apps. Depending on this hierarchy, virtualization can be categorized into three different ways: full virtualization, paravirtualization and hardware assisted virtualization.

Full virtualization

Full virtualization is using binary translation combined with direct execution techniques, which fully abstracts the guest OS from the underlying hardware by the virtualization layer. In such case, a completed simulation of the underlying hardware is provided. As the guest OS does not recognize that it has been virtualized, no modifications are needed. User apps will run on ring 3, guest OS will run on ring 1, and the hypervisor will run on ring 0 (Figure 2.8). Full virtualization serves the best isolation and security for the VMs, and it also simplifies migration and portability [34]. KVM is the full virtualization solution for Linux on x86 hardware containing virtualization extensions.

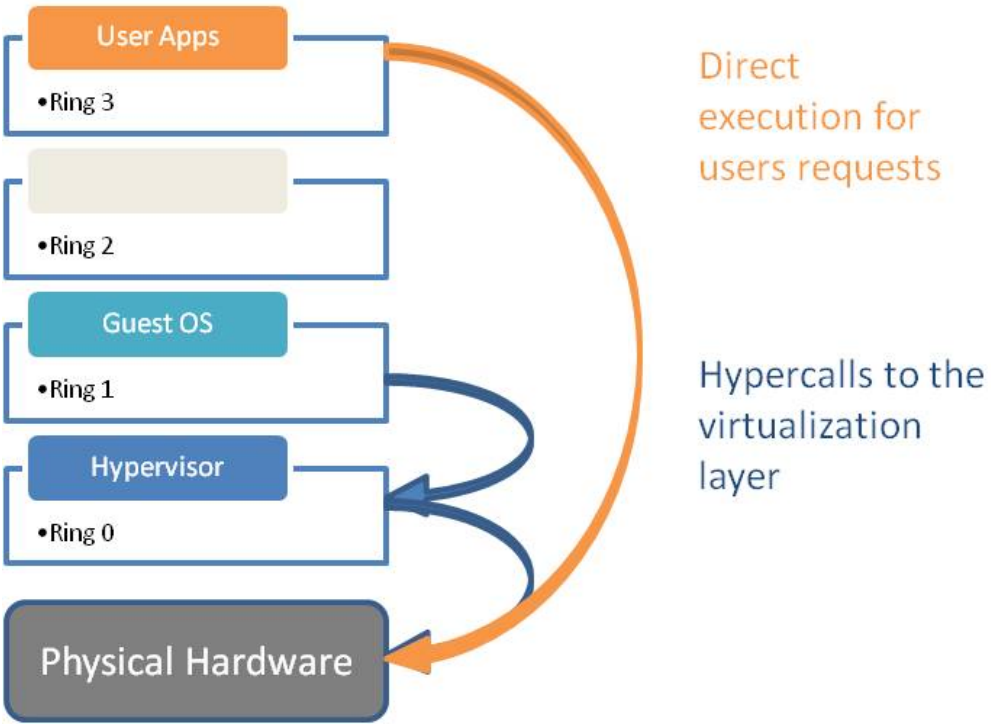

Figure 2.8 *Full virtualization*

Paravirtualization

Instead of the non-virtualizable instructions with hypercalls, paravirtualization involves the modified OS kernel which directly communicates with the virtualization layer hypervisor. It presents a software interface to VMs that is similar to the underlying hardware. As the OS has to be modified for this type of virtualization, its compatibility is poorer compared to the full virtualization. However, it is a relatively simple way of modifying the guest OS to enable paravirtualization. By this approach, user apps are still running on ring 3, while the guest OS is modified and located on ring 0, and a virtualization layer is coming below ring 0 (Figure 2.9) [34].

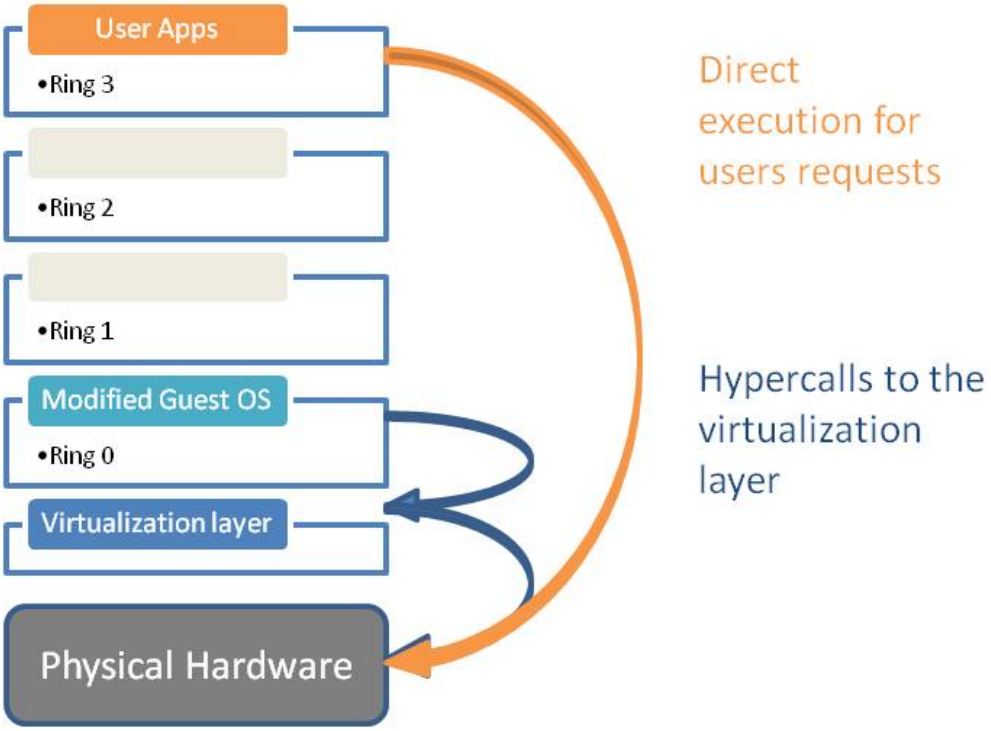

Figure 2.9 *Paravirtualization*

Hardware assisted virtualization

Hardware assisted virtualization is developed by hardware vendors to simplify the virtualization process. A new execution mode allows the hypervisor to run in a new root mode beneath ring 0. It enables efficient full virtualization using help from hardware capabilities. Privileged and sensitive calls are set to automatically trap to the hypervisor, like Figure 2.10 depicts [34]. This type of virtualization does not need the modification for the guest OS. It also enables a better performance.

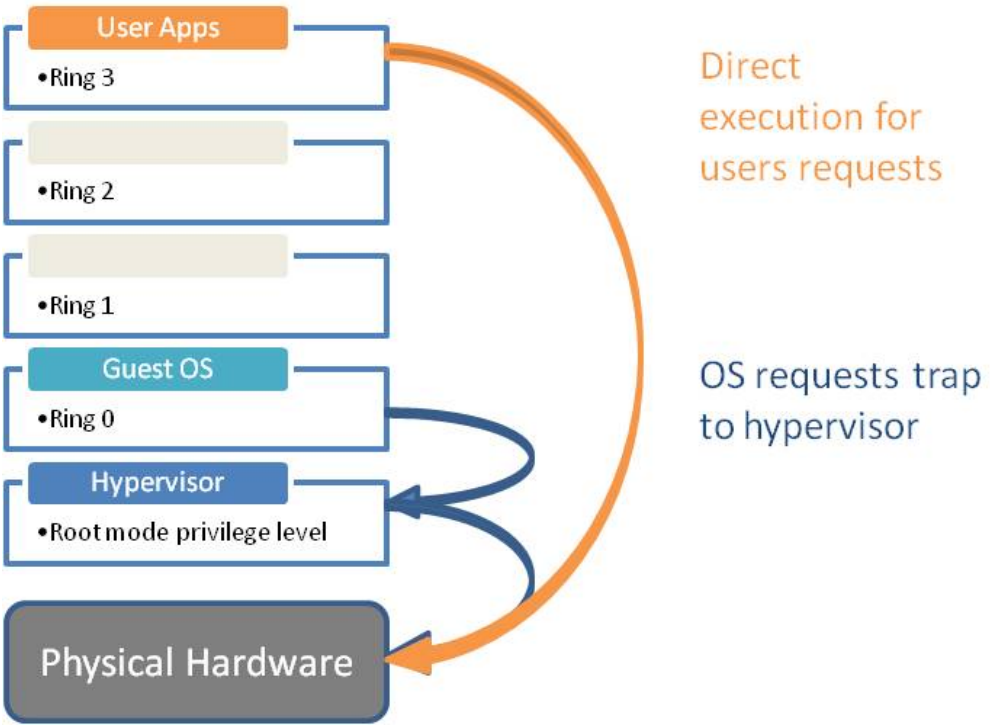

Figure 2.10 *Hardware assisted virtualization*

2.3 Cloudification

Virtualization is the way to create a shared resource pool, namely a cloud platform, from the physical hardware as we have already discussed. It is more related to the lower layer hardware. Whereas, cloudification is the process to migrate a traditional application from a local installation on the hardware to a web-based equivalent for the cloud. This clearly indicates that the object to be cloudified is the software which is running on the top layer. To make a real cloud based product, Figure 2.11 shows the process of the evolution from a legacy product to a cloud product.

A cloud product allows users to access it as a service (SaaS style), which is transformed to be used directly and customized on demand in the cloud. The most significant features of cloudification through traditional application programs are software virtualization and multi-tenancy design. Virtualized software allows application programs and data to be put into virtual layers instead of being put into basic file systems or registries. Multi-tenancy is the ability of the application program to provide the same service to different customers [37].

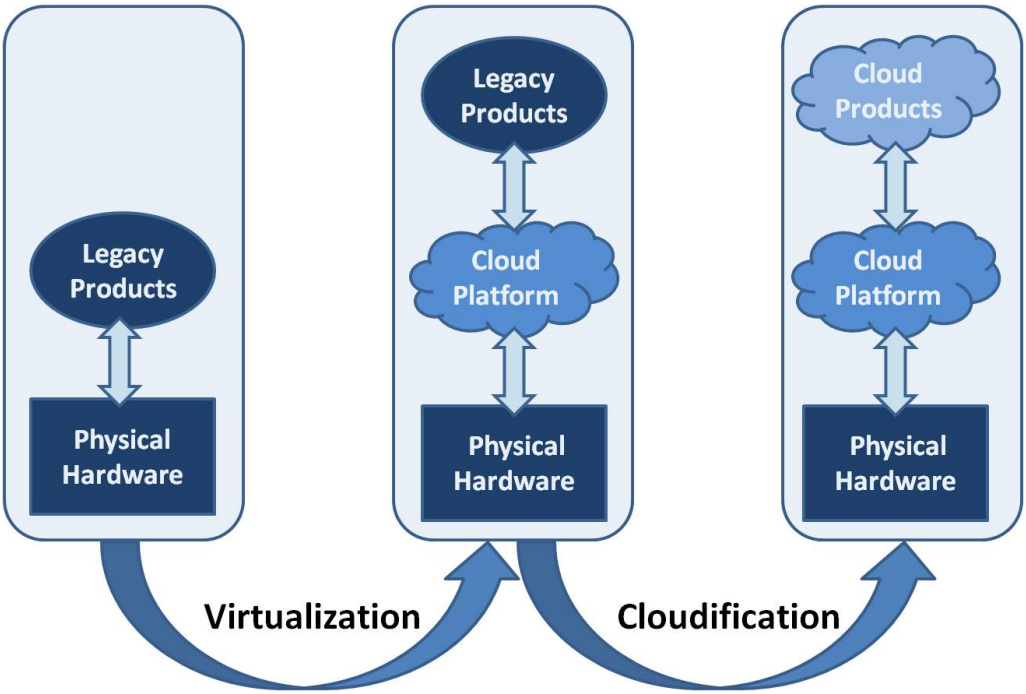

Figure 2.11 *Evolution from the legacy product to the cloud product*

2.3.1 Software virtualization

Software virtualization generates the virtual boundary that can separate and isolate various resources. Application programs are installed in the virtual layer and operated as normal. They do not have to suit the native OS, because they have been encapsulated from the underlying OS when they are being executed. Data in the application can be written in a specific destination and users are able to restore its initial state. Therefore, stable and flexible computing environments are guaranteed [8].

According to the definition from VMware [36], software virtualization is the ability to deploy software without modifying the host computer or making any changes to the local OS, file system, or registry. Using this technology, organizations can provision software through the company without installation conflicts, system changes, or any impact on stability or security. A virtualized software not only simplifies the complexity, but also supports delivering and accessing the traditional application deployment. It significantly shortens the time and regression testing needed for the successful delivery and update of the application. As the application is virtualized and exclusively runs in user mode, the host OS and other applications can be protected from the potential corruption. Figure 2.12 depicts how a virtualized application is being used by an end user.

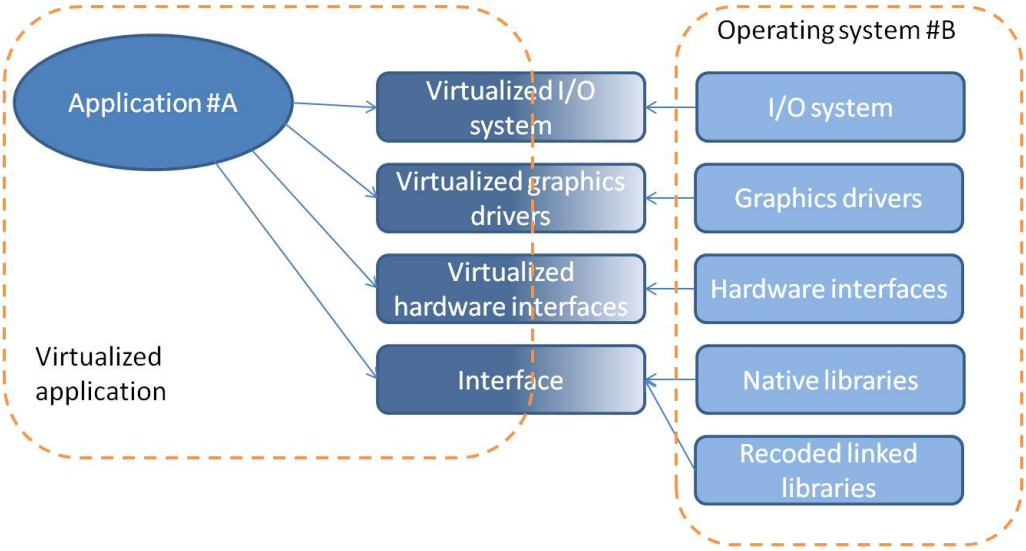

Figure 2.12 *Virtualized application*

2.3.2 Multi-tenancy

A more specific definition of multi-tenancy is explained as an approach to share an application instance among multiple tenants by providing every tenant a dedicated "share" of the instance, which is isolated from other shares with regard to performance and data privacy. A tenant here is a group of users who are sharing the same view of an application they are using. The view includes the data they access, the configuration, the user management, particular functionality and related nonfunctional properties [7].

Based on Microsoft Developer Network[10], there are three approaches to managing multi-data: Separate Databases; Shared Database, Separate Schemas; Shared Database, Shared Schema [3].

Separate Databases

Separate Databases is the simplest way to reach data isolation for tenants. All tenants share computing resources and application's codes on the same server, while each tenant has its own isolated set of data. Metadata associates each database with the right tenant, database security prevents tenants accessing from other tenants' data. By this approach, it is easy to extend the data model of the application to satisfy various individual requirements. In case of failure, restoring a tenant's

[10]https://msdn.microsoft.com/en-us/default.aspx

data is relatively not difficult. However, it leads to a higher costs for maintaining hardware devices and backing up the tenant's data. This approach is appropriate for customers who are willing pay extra for the added security and customizability.

Shared Database, Separate Schemas

As the name of this method suggests, multiple tenants are located in the same database, while each tenant has its own set of tables which are grouped into a schema. As the previous approach, this is comparatively simple to be implemented, and tenants can extend their data model easily. It is able to support a large number of tenants per database server. Applications can be purchased at relatively lower costs. However, it is complicated to restore a tenant's data in the case of failure, the database has to be restored to a temporary server, then tenants' tables are imported into a production server. This is a time-consuming task for the database administrator. Applications which use a small number of database tables and for customers who can accept their data co-located with other tenants are suitable for this method. In the future, the cloudified PCU will be designed to use this type to address the multi-tenancy issue.

Shared Database, Shared Schema

In the third way, the same database and the same set of tables are used to host multiple tenants' data. The tenant id is the key that associates records of an appropriate tenant in a given table. This approach has the lowest hardware and backup costs among the three methods, because it supports to serve the largest number of tenants in each database server. As multiple tenants share the same database server, additional development for security is required to ensure tenants' data. When restoring data for a tenant, much more effort is needed. Individual rows in the production database must be removed and reinserted from the temporary database. In case of a large number of rows in the affected tables, performance will suffer for all tenants in the database server. This approach is appropriate for customers who can give up data isolation in exchange for a lower cost.

3. GPRS NETWORKS

In this chapter, we will have an overview of the GPRS radio network. As the key unit in the GPRS radio access network as well as the being cloudified object in the thesis, PCU will be introduced in general afterwards.

3.1 Introduction to GPRS

GSM is also known as the second-generation (2G) digital cellular network used by mobile phones for two decades. It is a circuit-switched network, ideal for the delivery of voice. However, due to the limitations of sending packet data in GSM, GPRS was introduced to add packet-switched functionality and to support the delivery of the Internet for mobile devices. According to the European Telecommunications Standards Institute (ETSI) [1] technical report [33], the following requirements should be met by GPRS:

- Enable new and existing applications to be attached to GSM.
- Support both connectionless and connection oriented services.
- Offer a flexible service at low cost to the user.
- Use scarce network resources as efficiently as possible.

Based on the requirements stated above, the architecture of GPRS network was designed as Figure 3.1 illustrates. It was combined with the typical 2G network to form the 2.5G network, which was well accepted by people. Compared with the GSM data transmission, GPRS provides a higher throughput for data connection. In the rest of this section, network elements in the architecture will be described mostly according to the book [26].

[1] http://www.etsi.org/

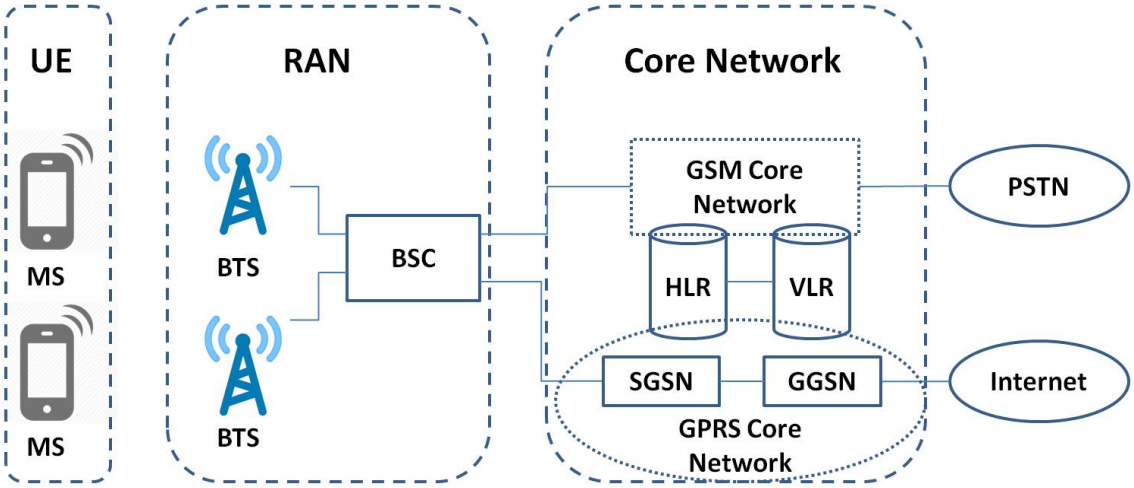

Figure 3.1 *GPRS network architecture*

3.1.1 User equipment (UE)

UE could be any devices equipped with a mobile broadband adapter which is used directly by an end user to communicate. A mobile station (MS) is the terminology in 2G system; it could be roughly considered as the UE in 3G and later systems. All the user's equipment and software needed for communication with a mobile network belong to MS.

3.1.2 Radio access network (RAN)

RAN is located between the UE and the core network, in which the Base Transceiver Station (BTS) and the Base Station Controller (BSC) are included.

BTS is the physical equipment center together with the base station equipment, which is to facilitate communication between the UE and a network. A BTS commonly has several transceivers that serve several different frequencies and different sectors of the cell. It is controlled by the BSC via the base station control function.

BSC is the network component with functions for controlling one or more BTSs. It is for handling traffic and signaling between mobile phones and the network switching subsystem. The BSC also allocates radio channels, receiving measurements from mobile phones, and controls handovers from BTS to BTS. It is the key element in the RAN, a BTS controller as well as a full switching center.

3.1.3 Core network

Core network is the last big part of the architecture, which is used to connect to different networks (circuit-switched network or packet-switched network) according to the coming traffic. GSM core network is for the typical connection between phones (phone calls) either external (Public Switched Telephone Network (PSTN)) or internal to the GSM network. Whereas, GPRS core network is used to transmit IP packets to external networks like the Internet.

Serving GPRS Support Node (SGSN) is the node to serve the MS. It knows the location of the MS by cell or routing area depending on the terminal's mobility management state. It is also responsible for the user's authentication and ciphering to guarantee the connection to be protected. In addition, SGSN can be used to collect required charging data on GPRS-connections by forming charging data records.

Gateway GPRS Support Node (GGSN) is the node for enabling data transmission between data networks and the GPRS network. It is switched to the SGSN via the IP based GPRS core network, and visible to external networks as a router of the IP network. Unlike the SGSN, GGSN could collect charging data to the external data networks.

Home Location Register (HLR) and Visitor Location Register (VLR) are shared by both the GSM core network and the GPRS core network. HLR is a database that contains the subscriber's data of the MS together with the location information. The data is for charging and enabling of the routing. VLR is the database for storing information regards to the MS roaming in its area. Exchanging between HLR and VLR will help VLR to get required data for handling calls to the MS.

3.2 PCU

In addition to the elements that are visible in the architecture, PCU is another key part to be introduced in the RAN area. It is a logical entity that processes packet data to and from the radio interface. Radio Link Control (RLC)/Media Access Control (MAC) functions are charged by PCU, for example segmentation and reassembly of Logical Link Control (LLC) frames, transferring RLC blocks, radio channel management, radio resource assignment and so forth. The BTS will receive communication with the PCU. As shown in the Figure 3.2, the PCU is located normally between the Abis interface and the Gb interface, working as a plug-in unit of the BSC. The Abis interface is an internal interface linking the BSC and the BTS,

which allows control of the radio equipment and radio frequency allocation in the BTS. The Gb interface connects the RAN and core network, which allows for the exchange of signaling information and user data.

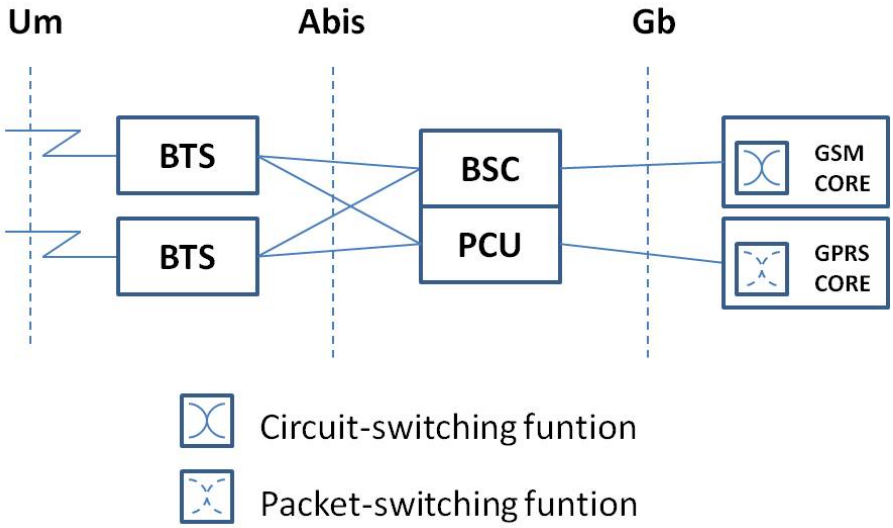

Figure 3.2 *PCU position*

From the user plane point of view, when PCU is attached to BSC, L1/L2 protocol between BTS and PCU is introduced. The protocol guarantees the transmission of RLC/MAC blocks from PCU to BTS (Figure 3.3), and allows in-band signaling for control of BTS from PCU and synchronization between each entity.

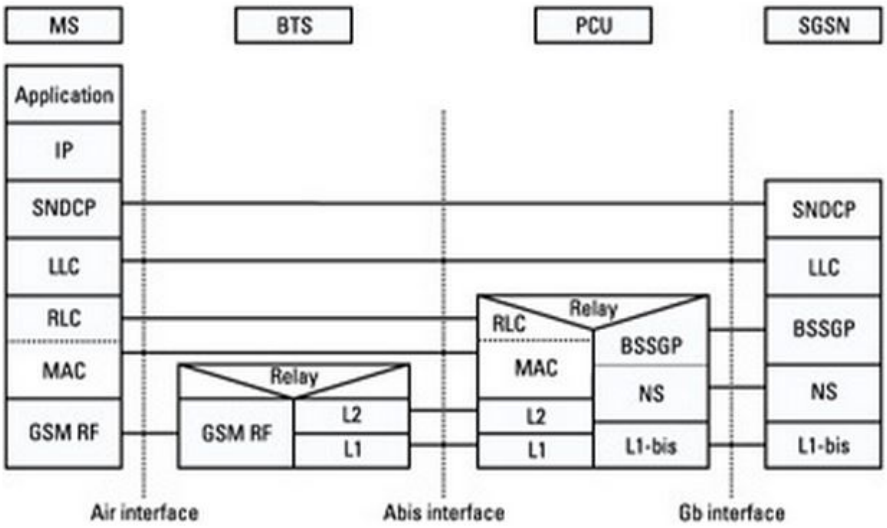

Figure 3.3 *Data transmission plane for PCU attached to BSC [32]*

- `GSM Radio Frequency (GSM RF)`: A physical layer protocol between MS and

BTS. It implements functions for channel coding, cell selection/reselection, timing advance and power settings.

- `RLC/MAC`: RLC provides a radio solution dependent reliable link; MAC controls the access signaling procedures for the radio channel.
- `LLC`: A reliable and secure logical link between MS and SGSN is guaranteed.
- `Sub Network Dependent Convergence Protocol (SNDCP)`: User data packets will be segmented into LLC packet data units here.
- `L1/L2`: Not a GPRS standard protocol, but can be self-defined by operators and vendors.
- `Network Services (NS)`: Base Station Subsystem GPRS Protocol (BSSGP) data units are carried by NS between BSC and SGSN.
- `BSSGP`: LLC frames and signaling between BSC and SGSN are carried. Both user and cell level of flow control for downlink direction are addressed here.

All interpretations for the transmission plane protocol stack are referenced from here [9].

4. OPENSTACK

The first part of this chapter introduces the background of OpenStack. At the following step, the deployment of OpenStack will be described to get the OpenStack cloud platform.

4.1 OpenStack services

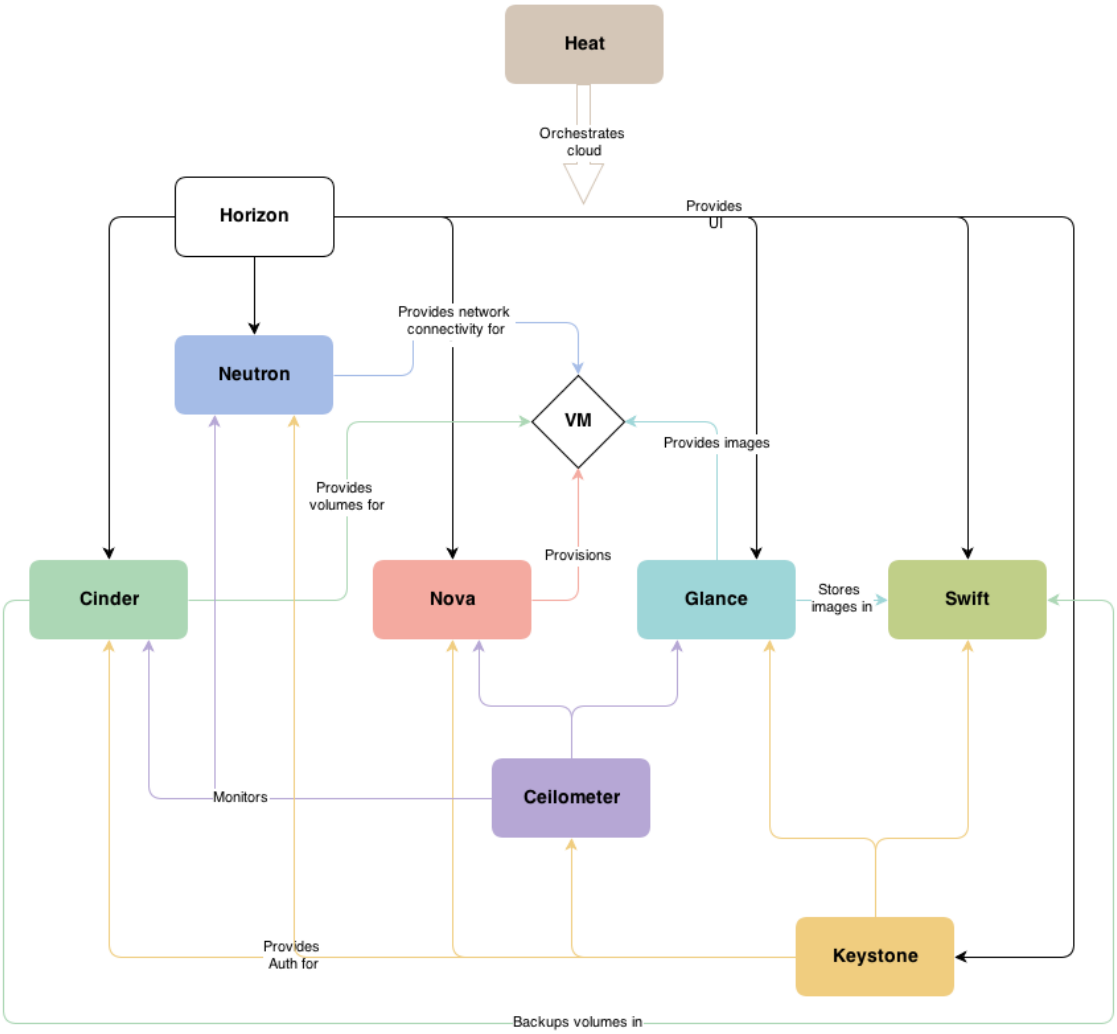

Figure 4.1 *OpenStack conceptual architecture[18]*

OpenStack is a free open-source cloud computing platform for creating private and public clouds. It manages a large pool of computing, storage and networking resources throughout a data center [22]. A series of interrelated services (Figure 4.1) provide an IaaS solution and are jointly supported by over 200 companies and organizations all over the world [16]. Every 6 months, there will be a new stable release for OpenStack. The frequent development milestones keep it always fresh. When the thesis is ongoing, OpenStack Juno is the recent stable release [23]. Therefore, OpenStack Juno is used in the thesis.

4.1.1 Compute service

`Nova` is the project name for the OpenStack compute service. It is the main part of an IaaS system that allows users to host and manage cloud computing systems. No virtualization software is included in nova. Instead, it defines drivers to communicate with the underlying virtualization mechanisms. Various hypervisors are controlled by nova through an API server. The API server makes command and control of the hypervisor, storage and networking programmatically available to users. All communications among components are through the message queue, which avoids blocking responses for each component. Nova is mainly used for managing instances that are running on the host machines. It will schedule the resources for instances to the available host by default, but users can also decide where the instance should be located. The resource to be allocated to the instance can be used from the default flavor list, and users are allowed to define their own customized flavor. The flavor in OpenStack means the size of the resources to be allocated to VMs, such as the number of cores, the size of memory, and so on. Instances can be launched, removed, resized or even migrated; all those services are served by nova. When users are going to have the remote console access to the guest VMs, nova also initiates the connection to the VNC console proxy. In addition, legacy networks are handled by nova, however nowadays, OpenStack networking service is also responsible for it [18].

4.1.2 Networking service

OpenStack networking service is recognized as `neutron` project, which allows users to define network connectivity and addressing in the cloud. It provides network, subnet, port and router object abstractions, like the network components in the physical world. Users can create virtual network topologies, even including firewalls and load balancer services. The software-defined network is responsible for connecting directly with each VM, which is also known as the internal network. However,

the external network represents a view of the physical network that can be accessed outside of the OpenStack cloud environment [20].

A standard networking deployment (Figure 4.2) well illustrates the structure of the network connectivity among multiple hosts [18]:

- Management network: Internal connectivity among OpenStack components are provided, only can be reached within the data center.
- Data network: Communication among VMs in the cloud is served.
- External network: Internet connections for VMs are supported.
- API network: All OpenStack APIs are exposed to the Internet to allow user access.

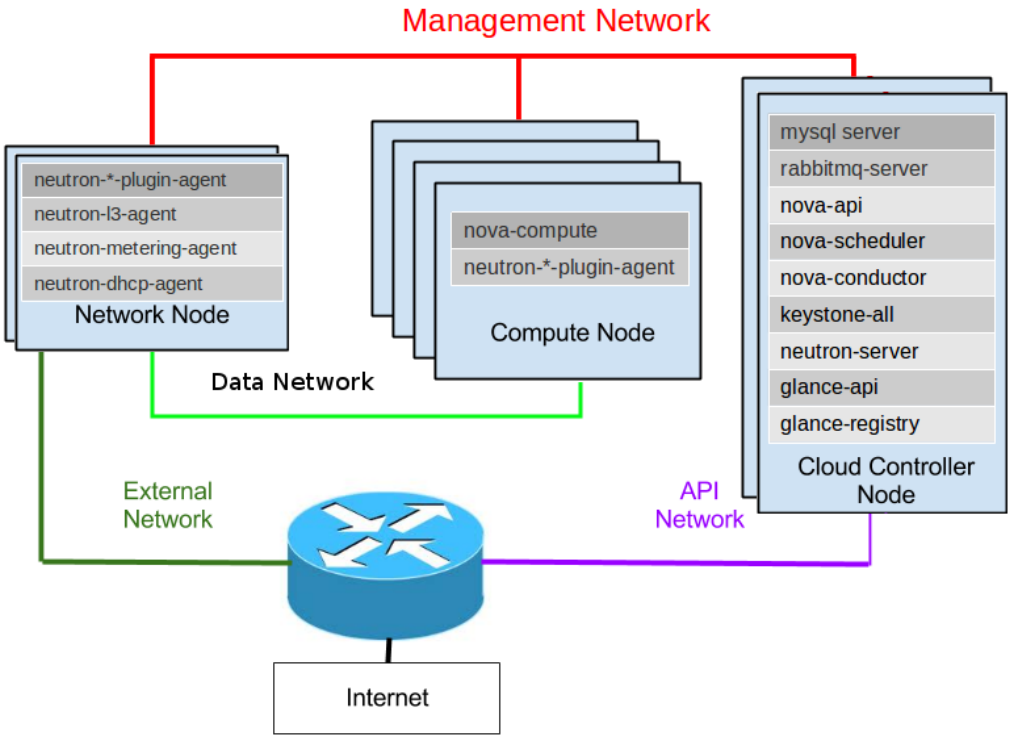

Figure 4.2 *Network connectivity among hosts [18]*

Different types of provider networks are supported by neutron [18]:

- `Flat`: All instances or even the host can share the same network.
- `Virtual local area network (VLAN)`: Users are allowed to create multiple providers by using different VLANs presented in the physical network.

- `Virtual extensible LAN (VXLAN) and Generic routing encapsulation-(GRE)`: A network overlay to support private connections among VMs, a router is needed to enable the communication to the outside network.
- `Local`: The local network to the compute host, which is isolated from any external networks.

Some plugins are introduced to OpenStack networking to offer a custom back-end implementation of the networking API:

- `Open virtual switch (OVS)`: The massive network automation through programmatic extension is enabled, the distribution across multiple physical servers is also supported as well [15].
- `Liunx bridge`: The sub-interfaces of VLANs on each host is enslaved [11].
- `Module layer 2 (ML2)`: OpenStack networking to simultaneously utilize the different layer 2 networking technologies is allowed [14].

4.1.3 Storage services

In OpenStack cloud services, there are two types of storage, object (`swift`) and block (`cinder`), that support a variety of provisioning options.

Object storage

Swift is an open source software for creating redundant, scalable data storage by using clusters of standardized servers to store petabytes of accessible data. It is a fully distributed storage system for static data without any central points to control, where the static data is scalable, redundant and durable. As objects are written to different hardware, data can be guaranteed to be replicated and integrated over clusters. Because OpenStack uses the software logic that data is replicated from one to another active point, no more expensive equipment is needed to avoid storage fail. This is a cost effective and scale-out way of storage [25].

Block storage

Cinder allows users to interact with block devices by attaching volumes persistently to the running VMs. Different storage platforms are supported for managing block

devices for servers. It is suitable for the performance sensitive scenario, such as database storage or expandable file systems. Powerful functionality to back up data that is stored on the block storage volumes are served by the snapshot management. The snapshot can be restored or created for a new block storage volume. This has better performance and integration for enterprise storage platforms [25].

4.1.4 Other services

Besides the main services that have been presented above, OpenStack also provides some other services that will help to build up and maintain the cloud platform. Those services will be introduced in the following sections.

Identity service

`Keystone` is known as OpenStack identity service. It provides central authentication and authorization for other OpenStack services. For example, when a user is going to access OpenStack services, the user has to go through the following steps [1], as depicted in Figure 4.3:

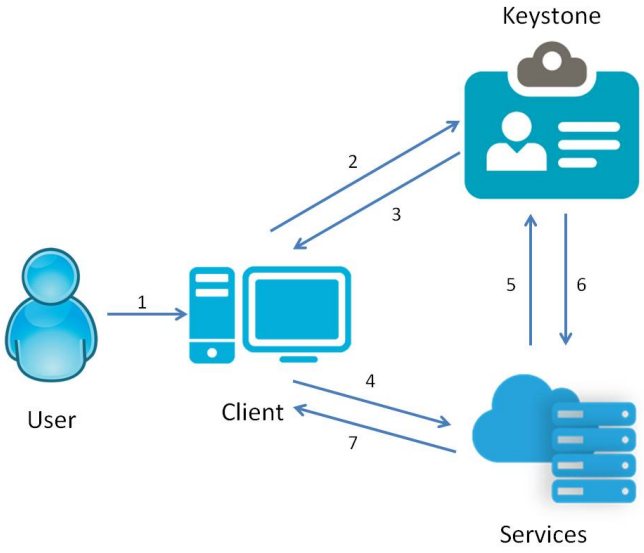

Figure 4.3 *OpenStack authentication and authorization*

1. The user should use the credential to enroll to the service client.
2. The client sends an authentication request along with the credential to the keystone server.

3. A scoped token and a list of endpoints to different services are sent back to the user.
4. The scoped token along with the user's request are sent to the service.
5. The scoped token is sent to the keystone server for validation.
6. Authenticate the user with the service.
7. The user's request is processed and the response is sent back to the client.

As the authorization and verification will be passed among all other OpenStack services to proceed them to be used, keystone should be the first service to be configured [5].

Image service

OpenStack image service is also called `glance`, which allows users to discover, register and retrieve VM images. API requests for disk or server images and image metadata from end users or OpenStack compute components are all accepted. Users can store VM images in different repository types, from simple file systems in the host machine to object-storage systems like OpenStack swift. By default, all images will be uploaded and stored in the host machine file back-end at */var/lib/glance/images/* [20]. The running instance can be stored and cataloged as a backup; the backup and the stored image can be used as a template. Thus, whenever a new instance is going to be launched, glance will provide the ready template to use. In fact, glance never stores any images, but rather catalogs them and provides metadata from a storage back-end data store. Then, other modules should communicate with glance to fetch the image metadata.

VM images are the key objects that are uploaded and managed by glance. There are various disk formats for VM images, like `raw`, `qcow2`, and so on. The image format of raw is the simplest one and natively supported by both KVM and Xen[1] hypervisors. The raw image can be thought as the bit-equivalent block device file. The format of qcow2 stands for QEMU copy-on-write version 2, which is commonly used by the KVM hypervisor. It has smaller size and supports for snapshots. Therefore, this is often used as the image format in OpenStack [21].

[1] http://www.xenproject.org/

Dashboard

OpenStack dashboard (`horizon`) is a web based graphical interface that can be accessed by users and administrators. Through the dashboard, users can easily deploy, automate, or monitor their cloud-based resources. It is designed to be extensible with additional plugins. Thus, users could make customized management by their own needs [17]. An example of the dashboard is illustrated in Figure 4.4.

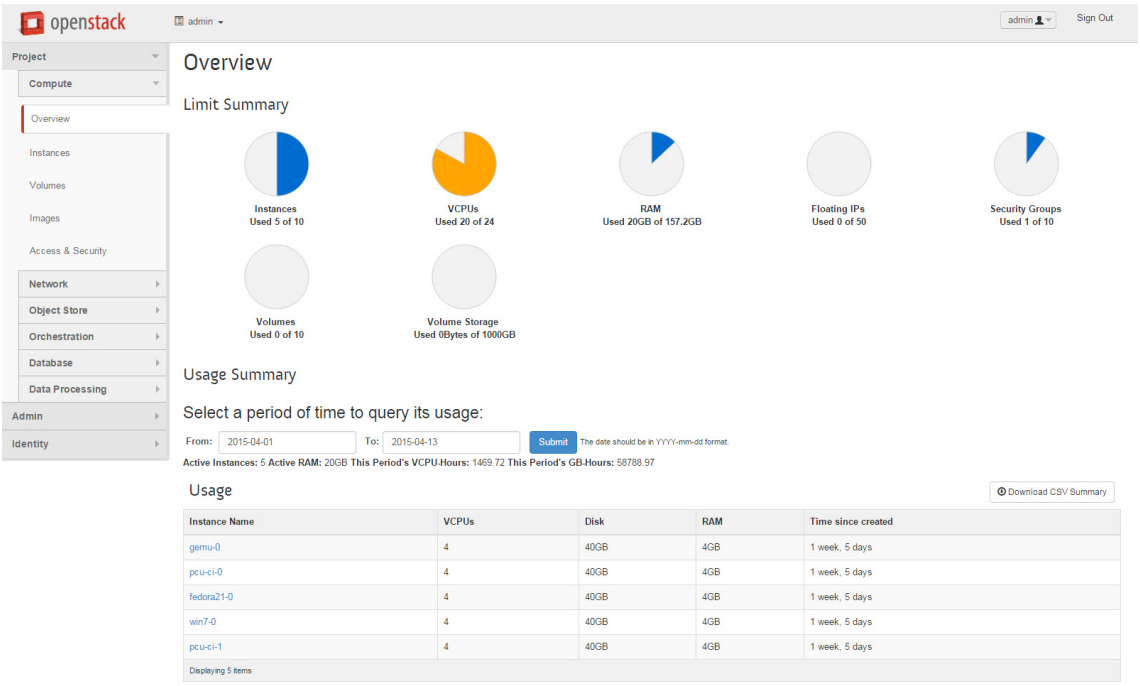

Figure 4.4 *OpenStack dashboard overview*

Telemetry service

Telemetry service (`ceilometer`) in OpenStack was created originally for billing systems for OpenStack cloud resources. It covers the required processing metering portion for billing, where information used for billing is collected. Later, it has grown gradually to support other purposes rather than just billing. Event notifications triggered by different executions in OpenStack would be recorded, and the alarm system can be set according to users' requirements [18].

Orchestration service

OpenStack orchestration service (`heat`) allows users to orchestrate clouds, meaning that resources in stacks can be automatically configured and deployed. A group of virtual elements, such as servers, networks, can be automatically scaled in the OpenStack cloud. Orchestration stacks are defined in templates, where resources, parameters, inputs, constraints and dependencies are described in the document. As it uses the notation which loosely follows Python/Ruby structural conventions, it is easy to be created and utilized with source-code management systems [18].

In addition to the services that have been introduced, OpenStack also supports database service such as `trove` and data processing service such as `sahara` nowadays. Moreover, it will support more additional services, for example bare metal (`ironic`), queue service (`zaqar`), and so on, in the future release [24].

4.2 OpenStack deployment

4.2.1 Preparation for OpenStack deployment

In this thesis, CentOS 7[2] is selected as the OS on the host server. CentOS Linux is a popular community-supported distribution derived from the open source code of Red Hat Enterprise Linux (RHEL)[3]. In another word, CentOS Linux is functionally compatible with RHEL. It is appropriate for open-sourced projects, for example it has enterprise grade supports for OpenStack Platform deployment [30]. CentOS 7 is the latest Linux version for CentOS at the time when this thesis is written. In addition, the hardware server should have at least 2GB RAM; processors should be extensible for virtualization; and at least one network adapter is needed. This is the minimal hardware requirement [29].

Before configuring the host OS, one important thing is to manually create the OS partition. Thus, we have an idea of which directory shall be used for the OpenStack deployment. This is because OpenStack will recognize the space under that directory as the storage to be used by default. Therefore, it is better to have large available space for OpenStack. Otherwise, the storage in OpenStack will be the bottleneck for the cloud environment. For the host OS installation, the minimal installation should be enough for deploying an OpenStack based cloud. With the minimal installation, it will take the minimum time to complete the OS installation and update. More

[2]http://www.centos.org/
[3]http://www.redhat.com/en/technologies/linux-platforms/enterprise-linux

importantly, the minimal installation will minimize the resource usage by the host OS, and maximize the available resource for the cloud. The limitation of that is there is only the command line console interacted with the user, and no graphical interface at all. Because there will be some other tasks to be handled with the host OS, to get it working in a more user-friendly way, the GNOME type installation can be chosen. It provides users a graphical interface, which is more convenient for starting other tasks. For the network, it is better to use a static Internet Protocol (IP) address instead of the Dynamic Host Configuration Protocol (DHCP) mode. The reason is that the host IP will be used for OpenStack as the service entrance later on. With a manually assigned static IP address, it will give the guarantee for acquiring OpenStack services. Besides, the host name can be defined in the network configuration. By clearly defining a host name, it will help to distinguish different nodes for the multi-node OpenStack deployment. Now it is the time to start the host OS installation.

After the host OS installation, the first thing is to make sure whether we need a proxy or not. If yes, the proxy should be added in */etc/yum.conf* in the form of `proxy=http://<proxy_ip>:8080`. A local valid proxy IP address should be used to replace the `<proxy_ip>` part.

Now, we should use the following command to get the host OS updated.

```
yum -y update
```

This will take some time, especially for the installation rather than minimal installation. `yum` is the command to manage software in RHEL based Linux distribution. With `-y` option, it will automatically select the "yes" option whenever it is prompted during the execution. Thus, users do not need to give a yes response every time when it is needed. When the host OS update is completed, a reboot of the system is necessary to complete the update.

If the system is using `NetworkManager`, we should disable it to get OpenStack working properly in the later phase, because OpenStack cannot work with `NetworkManager` by default. To disable and stop the `NetworkManager` service in the system, follow the commands below. The last command shows the status of `NetworkManager`.

```
systemctl stop NetworkManager
systemctl disable NetworkManager
systemctl status NetworkManager
```

To totally avoid the interference caused by `NetworkManager`, we should add one line to the network interface configuration file which is located at */etc/sysconfig/network-*

scripts/. Generally speaking, the system should have at least two network interfaces for use by OpenStack, one is reserved for the Internet connection, the other one is reserved for the OVS bridge. Append `NM_CONTROLLED=no` to both of the network interface configuration file. To get the configuration activated, restart the network.

```
systemctl restart network
```

As OpenStack has its own firewall service, it may conflict with a system-level firewall. Therefore, it is advised to stop and disable the firewall of the system in the proof-of-concept phase. However, when there is a product, the host firewall should be set up and negotiable with OpenStack for more security. Execute the commands below to get this done and to check the status.

```
systemctl stop firewalld
systemctl disable firewalld
systemctl status firewalld
```

To install the latest OpenStack packages, some configurations need to be done first according to the official guide from OpenStack Juno version[20]. The *yum-plugin-priorities* package allows the assignment of relative priorities to the configured software repositories. It is used by the RDO[4] release packages. RDO is a community of people for using and deploying OpenStack on RHEL. To install the package, follow the command below.

```
yum -y install yum-plugin-priorities
```

Extra Packages for Enterprise Linux is useful for creating, maintaining, and managing a high quality set of additional packages for Enterprise Linux. To configure that, execute the command below.

```
yum -y install epel-release
```

To enable the RDO repository, download and install the latest RDO released OpenStack package, by executing the following command.

```
yum -y install http://rdo.fedorapeople.org/rdo-release.rpm
```

Due to RHEL and CentOS enable `SELinux` (a mandatory access control security mechanism implemented in the kernel [6]) by default. Install the `openstack-selinux` package can help to automatically manage the security policy for OpenStack services.

[4]https://www.rdoproject.org/Main_Page

```
yum -y install openstack-selinux
```

Instead of using `openstack-selinux` package to manage the security policy, manually set the `SELINUX` level from `enforcing` to `permissive` can also solve the potential issue. Firstly, modify `SELINUX=enforcing` to `SELINUX=permissive` in the configuration file */etc/selinux/config*. This will guarantee SELINUX to be kept at the permissive level even the system is rebooted. Then, to apply the permissive level into the current situation, use the following command.

```
setenforce 0
```

By this phase, we should be ready to start the OpenStack deployment. However, some extra packages could be installed, for instance, *vnc-server* could be a very useful tool used for the remote connection. But here we will skip it, as people will have different requirements based on their needs.

4.2.2 Single node OpenStack deployment

OpenStack can be deployed manually according to the official installation guide [20]. This allows us to get familiar with the OpenStack configuration from the scratch step by step. However, by following the official guide and configuring everything from the beginning, it really takes time to get the work finished. Therefore, some automatic OpenStack deployment frameworks can be taken into consideration, which make the OpenStack deployment much easier to be done. One of the famous OpenStack deployment framework is called `Packstack`. It is used for deploying different parts of OpenStack on a single or multiple pre-installed servers over `Secure Shell` (SSH) automatically [12]. To get `Packstack` installed in the system, follow the command below.

```
yum -y install openstack-packstack
```

With a quick start of OpenStack deployment in a single node, we execute the following command.

```
openstack --allinone
```

However, if we are going to deploy a customized OpenStack, it is better to handle the related configuration in our own hand. By running the following command, we could create an answer file, which contains numerous configuration references for the OpenStack deployment.

```
openstack --gen-answer-file <name_of_the_file>
```

Open the generated answer file and make the modification according to our own requirements. In case of expecting an external network for OpenStack, `CONFIG_NEUTRON_OVS_BRIDGE_MAPPINGS` and `CONFIG_NEUTRON_OVS_BRIDGE_IFACES` parts need to be taken care of [28]. Otherwise, networks established on OpenStack will be only internally valid. Detailed configuration of the answer file for the thesis project is in appendix A. Now, it is the time to make the OpenStack deployment. By the following command, `Packstack` will automatically deploy OpenStack based on the configuration from the answer file.

```
openstack --answer-file=<the_answer_file>
```

This will take some time to get the deployment done. If an error occurs during the deployment, the RDO Workarounds[5] may help. RDO Workarounds collects some common failed experiences and possible solutions for the OpenStack deployment. Also, by the hint of errors which should be printed on the console, we could follow the detailed log to locate the possible cause. Just keep in mind, next time when we run the `Packstack` automatic deployment for OpenStack, the same answer file should be used. The reason is that all password settings had been utilized during the last deployment. Therefore, if we recreate another answer file to deploy OpenStack, it might fail once again (passwords in answer file are generated randomly each time when the answer file is created).

Once the deployment is successfully completed, it should look like Figure 4.5. The Uniform Resource Locator (URL) address can be seen from the red circle part, which is used for the OpenStack dashboard. The login account information can be found from a file called `keystonerc_admin`, which should be generated automatically when OpenStack is successfully deployed. By now, a plain OpenStack based cloud for a single physical server has been created successfully.

Note, the deployment might fail due to a failure of `mongodb` when using the version `RDO_juno_openstack_packstack_dev_1462`. Because there will be two configuration files for `mongodb`, one is generated by `puppet` at */etc/mongodb.conf*, the other one is generated by `mongodb` itself at */etc/mongod.conf*. To solve the problem, we need to modify the source code of `puppet` at */usr/share/openstack-puppet/modules/mongdb/manifests/params.pp*. In the file, find the OS family that we are using, and modify the value of *$config* from */etc/mongodb.conf* to */etc/mongod.conf*. Now, rerun the RDO scripts, the problem should not be there anymore.

[5] https://www.rdoproject.org/Workarounds

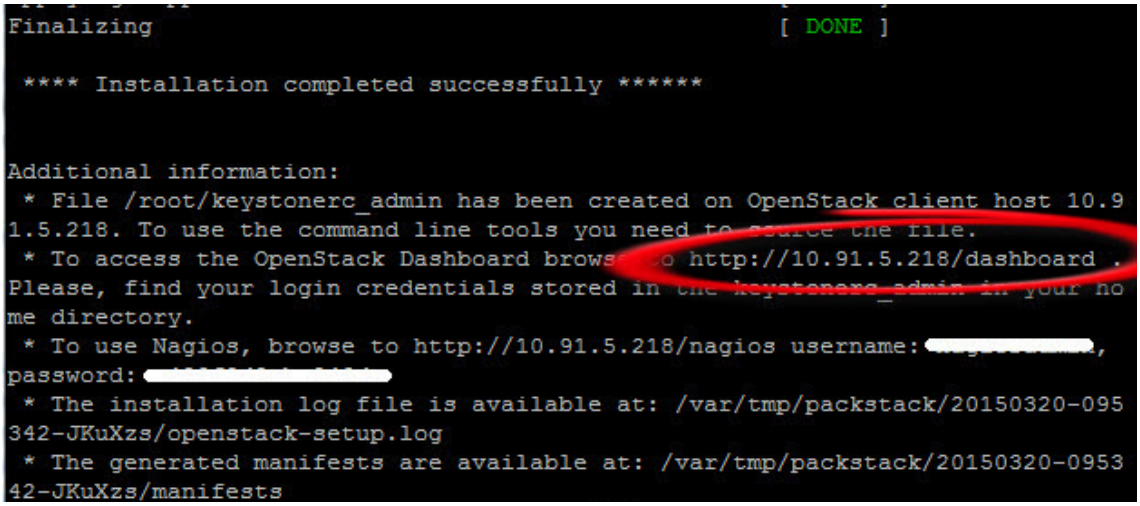

Figure 4.5 *OpenStack deployment completed*

4.2.3 Multi-node OpenStack deployment

Previously, we discussed about the single node OpenStack deployment module. Single node OpenStack deployment is the easiest way to get started. However, when the single server is not powerful enough, we might consider including an extra server to provide more resources, or even deploying an OpenStack with several servers from the start. In this section, we will introduce the multi-node OpenStack deployment.

One way to get a multi-node type of OpenStack deployment is to add extra nodes to the previously deployed OpenStack. For instance, if we are going to deploy another compute node to balance the server workload, we could modify the answer file that was used in the last successful deployment. In the answer file, we can change the value of `CONFIG_COMPUTE_HOSTS` from the value of the first host IP address to the host IP address which is going to be appended. In addition, `CONFIG_NEUTRON_OVS_TUNNEL_IF=<public_IP>` is used for setting up an OVS tunnel. It allows VMs in different physical networking hosts to have the encapsulated network connection. Moreover, by adding `EXCLUDE_SERVERS=<serverIP>` to the answer file, the server that has been configured previously will not be reconfigured this time. After the answer file has been modified, use `Packstack` framework to deploy the new node to attach to the previous OpenStack [27]. When the deployment is completed, from the OpenStack Dashboard, we could observe the new node from the hypervisor menu as Figure 4.6 depicts. This is the two-node-module (two real physical servers joined) system information for the OpenStack based cloud.

Another way to get a multi-node OpenStack based cloud is to deploy OpenStack in multi-node type at the beginning. Like the two-compute-node module previously,

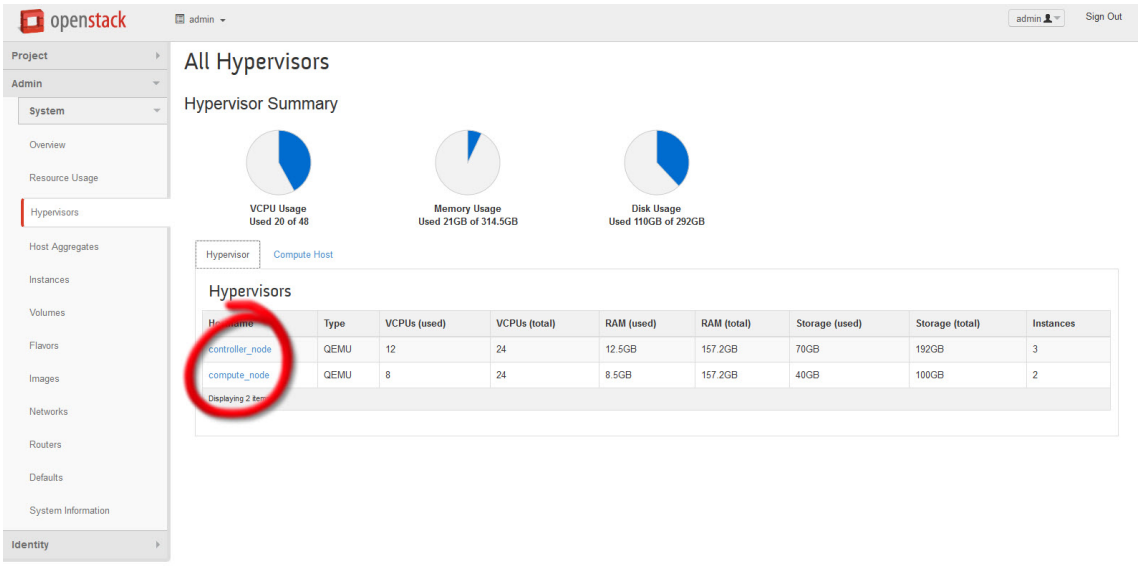

Figure 4.6 *Two-node-module hypervisors*

before the first time deployment, edit the answer file by giving both server's IP addresses to `CONFIG_COMPUTE_HOSTS` part. The two IP addresses should be separated by a comma. By running `Packstack` framework as before, we should get the same result as Figure 4.6 illustrates.

Thus, when it comes to more nodes than two, following the similar way of the two-node-module, we could establish the multi-node based OpenStack cloud. The multi-node based OpenStack cloud should have more powerful resources than a single node. Workload among different nodes will be balanced more or less equally by default. Nevertheless, it also supports allocating resources to a dedicated server. The multi-node OpenStack deployment is not that hard to do. If we have enough resources to use, the multi-node module is a good choice to get the cloud more flexible and powerful. As the situation might be different according to individual cases, this thesis will mainly focus on the general case of the single node infrastructure for the proof-of-concept.

4.2.4 Post deployment

After the plain OpenStack has been deployed successfully, some configurations need to be considered to get OpenStack cloud platform working correctly later on.

As we do not want to use any firewalls from compute and networking services, set `firewall_driver=nova.virt.firewall.NoopFirewallDriver` in */etc/nova/nova.conf* and `firewall_driver=neutron.agent.firewall.NoopFirewallDriver`

in */etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini*. In the same file of *ovs_neutron_plugin.ini*, also put `enable_security_group=False`. Thus, we will get rid of the firewall control. Otherwise, due to the conflicts among firewall settings, some errors might be encountered [20].

Compute service uses a metadata service to retrieve instance-specific data for VMs. Instances will access the metadata service at *http://169.254.169.254* when they are booting up. The URL is used for cloud computing platforms to distribute metadata to cloud instances. However, by using the flat/vlan type of network, the connection to metadata service will fail by default. Setting `enable_isolated_metadata=True` in */etc/neutron/dhcp_agent.ini* will assist instances to get metadata support on the isolated networks.

To totally ensure that there is no more firewalls and security issues blocking OpenStack services, it is better to disable and stop the `iptables` service in the system level. After that, check the status of the iptables service to make sure that it is really disabled and stopped.

```
systemctl stop iptables
systemctl disable iptables
systemctl status iptables
```

By default, the resource quota in OpenStack is not the real reflection of the physical resource. To get the one-to-one mapping of physical resources in OpenStack, we could use the following command. However, before we are going to do any configurations for OpenStack, we should first authenticate through keystone, otherwise, all command executions related to OpenStack will be denied. The first line is used for authentication, and the command starting from the second line is the resource quota update in general. The argument for `instances` is the new value of instances quota; `cores` is the new value of cores quota; `ram` is the new value of memory quota, but in the unit of megabytes (MB); `class` is the name of quota class to be set for.

```
source keystonerc_admin
nova quota-class-update [--instances <instances>]
  [--cores <cores>] [--ram <ram>] <class>
```

Because we have modified several configurations for OpenStack services, a restart for OpenStack services is necessary to get all modifications applied.

```
openstack-service restart
```

The configuration after OpenStack is deployed is ready by this phase. In the next chapter, we will discuss how PCU is deployed in the OpenStack based cloud platform.

5. PCU CLOUDIFICATION

This chapter explains how PCU is used based on the OpenStack cloud environment. To get a cloud based PCU, the cloud based testing environment for PCU will be established by either the manual deployment or the automated deployment. Then the instance management in the cloud will be discussed after the deployment.

5.1 Cloud based PCU testing environment

As the OpenStack cloud platform has been established, we should start to set up the cloud based PCU testing environment on top of OpenStack. OpenStack provides several ways to do that. One is to use the OpenStack web-based dashboard, which is the most user-friendly way. Commonly used arguments can be found through the dashboard. The other one is to use the command-line tools to set up the testing environment. By this way, the deployment will be more precise, as we could give some extra arguments that the dashboard cannot provide. Through the command-line tools, we could manually deploy required elements to OpenStack one by one, or by using a heat template to deploy all at once. In the thesis, we will mainly concentrate on the method of using command-line tools in the two different ways.

Besides, according to the resource that we have, the testing environment architecture is defined as Figure 5.1 illustrates. In the red box, we will have the PCU testing environment, where there will be a PCU instance, a configurator instance for configuring and monitoring the PCU instance, and a traffic generator used to generate random traffic to feed PCU for testing. In the yellow box, it is the PCU Continuous Integration (PCU-CI) testing environment, in which two PCU-CI instances are used for the PCU unit testing, as well as a configurator instance and a traffic generator. The PCU-CI instances are the merging of all working copies with a shared mainline several times per day. The configurator instance is better to concentrate on a single testing environment, thus we have two. While the traffic generator can be shared by both cases, hence we have only one. All instances will be connected to an external network to get the Internet access. Whereas, each testing environment will have its own internal network to avoid the interference from outside.

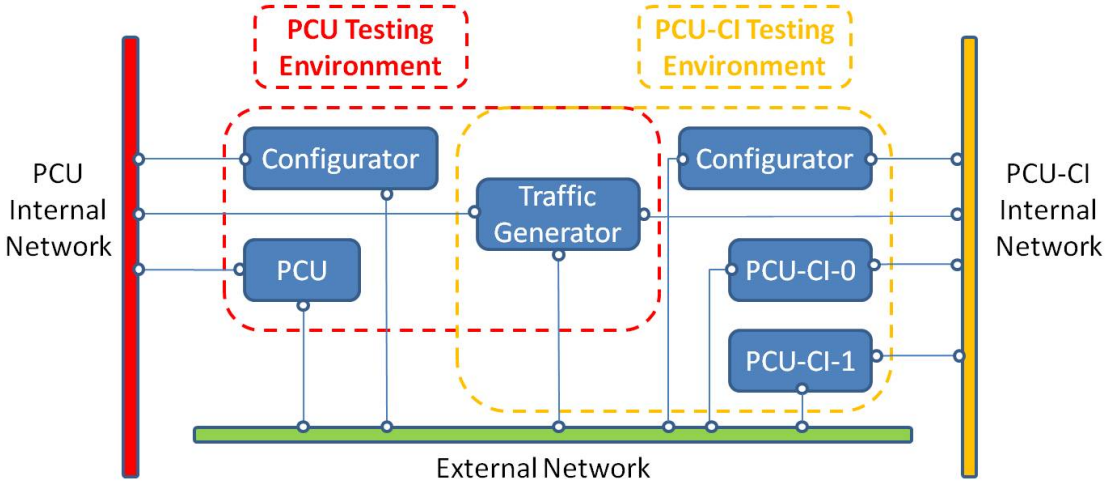

Figure 5.1 *Target scheme of the testing environment*

5.1.1 Manual deployment

Each OpenStack project provides a command-line client to enable people to access the project API through easy-to-use commands. Specifications of the command-line can be found from this reference [19].

Creating networks

We could start with the network topology. To get the Internet access from instances, we should first set up a network connected to the external interface with the following command. The argument `name` is the name of the network to be created. The `physical_network_name` is the name of the physical network over which the virtual network is implemented. The `network_type` is the physical mechanism by which the virtual network is implemented. For the proof-of-concept, flat mode network is the easiest way to do.

```
neutron net-create <name> [--provider:physical_network
  <physical_network_name>] [--provider:network_type
  <network_type>]
```

Compared to the external network establishment, it is much easier for the internal network creating. The internal network does not require options like physical providers. To create an internal network, a network name is all that is needed.

Like a physical network, a subnet should be assigned to the virtual network, where the exclusive range of IP addresses need to be specified to avoid conflicts with other

devices. In the command, the `name` is the name of the subnet to be created. The `gateway_ip` is the gateway IP address for this subnet. The `IP1` and the `IP2` are the start IP address and the end IP address that will be allocated for the subnet, respectively. The DHCP function of the subnet can be disabled or enabled. The network id or name to indicate which network this subnet belongs to should be indicated. At last, set the domain of the network for the subnet. All IP addresses belong to the external subnet range should be all valid and physically connected to the host machine. For the internal subnet range, they could be self-defined.

```
neutron subnet-create [--name <name>] [--gateway <gateway_ip>]
  [--allocation-pool <start=IP1,end=IP2>] [--disable-dhcp |
  --enable-dhcp] <network> <CIRD>
```

The reason for disabling the DHCP function for the subnet we just created is that we can get the full control of the IP address allocation. Also based on this, we should manually create the port to exactly allocate IP addresses to the instance. The command below shows how a port is created. The `name` is the name of this port. The `IP` is the desired IP address for the port. The `network` is the id or name of the network where the port should belong to.

```
neutron port-create [--name <name>] [--fixed-ip
  <ip_address=IP>] <network>
```

Now, we should have created all network elements that are needed. By using the following command, we are able to check the list of networks, subnets or ports that we have already created in OpenStack.

```
neutron net-list
neutron subnet-list
neutron port-list
```

To get more detailed information about a certain network, subnet or port, execute the command below. The argument of each command is the name or id of the certain element that is going to be checked.

```
neutron net-show <network>
neutron subnet-show <subnet>
neutron port-show <port>
```

Creating images

The prerequisite for launching instances in OpenStack is to have the proper images at first. To create and upload images to OpenStack, follow the command below. The `name` represents the name of the image that will be created. The `disk_format` is usually defined as `qcow2`. The `container_format` is usually defined as `bare`. The `disk_GB` is the minimum disk space in gigabytes (GB) needed to boot the image. The `disk_ram` is the minimum amount of ram in megabytes (MB) needed to boot the image. The `file` indicates the local file that contains the disk image to be uploaded. The option `is-public` decides whether the image is public to all users with access to the current project or not. The `is-protected` guarantees that only users with permissions can delete the image. The `property` can be arbitrary that associates with the image, e.g. `hw_vif_model=e1000` indicates the network interface card device model that will be supported by Intel DPDK[1] (a set of packet for fast packet processing). The `progress` option can show the uploading progress.

```
glance image-create [--name <name>] [--disk-format
  <disk_format>] [--container-format <container_format>]
  [--min-disk <disk_GB>] [--min-ram <disk_ram>] [--file
  <file>] [--is-public <True | False>] [--is-protected
  <True | False>] [--property <key=value>] [--progress]
```

According to the architecture (Figure 5.1), we will have at least 4 different images. They can be listed in general by the first command below, and also can be checked in details for a certain one by the second command, where the argument should be the name or id of the image.

```
glance image-list
glance image-show <image>
```

Creating instances

Finally, it is the time to deploy real instances to the cloud. Based on different resources that are required from images, the flavor for different images can be various. By default, OpenStack has provided a group of different flavors that can be directly used when launching a new instance. To check the flavor list, execute the command below.

```
nova flavor-list
```

[1] http://dpdk.org/

If there is no good candidate for the instance to be launched, it is also available to create our own flavor. To do so, follow the command below. The `name` is the name of the new flavor. The `id` can be defined by the user. The `ram` is the memory size in (MB). The `disk` is the disk size in (GB). The `vcpus` indicates the number of virtual central processing units.

```
nova flavor-create <name> <id> <ram> <disk> <vcpus>
```

After we guaranteed that images, flavors and network elements are all ready, we can now start to launch instances in OpenStack. The following command is used to boot up an instance in OpenStack. The `flavor` should be filled with the id or name of the flavor for the instance. Similarly for the `image`, a name or id of the image is expected. A `port-id` is allocated when the instance is booting up. This part can be multiple used in the same command when multiple interfaces will be allocated for the instance. The `poll` option can show the booting progress until it completes. The `name` is the name for the new coming instance. The option of the `port-id` can also be replaced by the `fixed-ip` with an IP address. As we have created those ports, we prefer to use the `port-id` to exactly allocate interfaces to the instance.

```
nova boot [--flavor <flavor>] [--image <image>]
  [--nic <port-id=<port-id>>] ([--nic <port-id=<port-id>>])
  [--poll] <name>
```

As before, to check the list of launched instances or even get more detailed information about one certain instance, the following commands can be used. The `server` indicates the name or id of the instance that is going to be checked.

```
nova list
nova show <server>
```

Because we disabled the DHCP function previously, the new instance probably cannot fetch the IP automatically by itself. Therefore, after the instance is launched and running in OpenStack, we could go through the dashboard to open the console of the instance to manually configure the associated IP for it. Besides, whenever the console access to the instance is needed, this is always a way to reach there. The entrance to the console of the instance can be found from Figure 5.2. Nevertheless, after the network connection is established for the instance, by the connection of SSH, the instance is also reachable.

When everything we need has been deployed successfully to OpenStack, from OpenStack dashboard, the network topology should look like Figure 5.3. All instances can be accessed through SSH or remote desktop connection from outside of the cloud

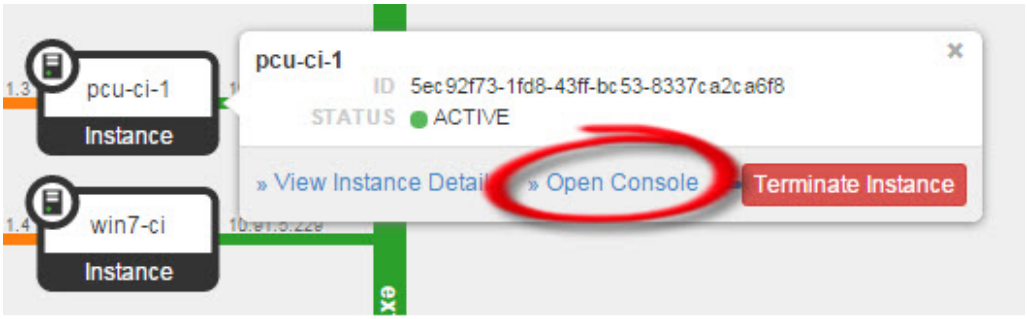

Figure 5.2 *Access to the instance through OpenStack dashboard*

via the interface connected to the `ext_net` (green). Both PCU and PCU-CI have their own testing networks, the `int-pcu-net` (blue) and `int-pcu-ci-net` (orange) respectively. The `win7-*` instances are the configurator instances, and the `gemu` instance is the traffic generator instance in the topology.

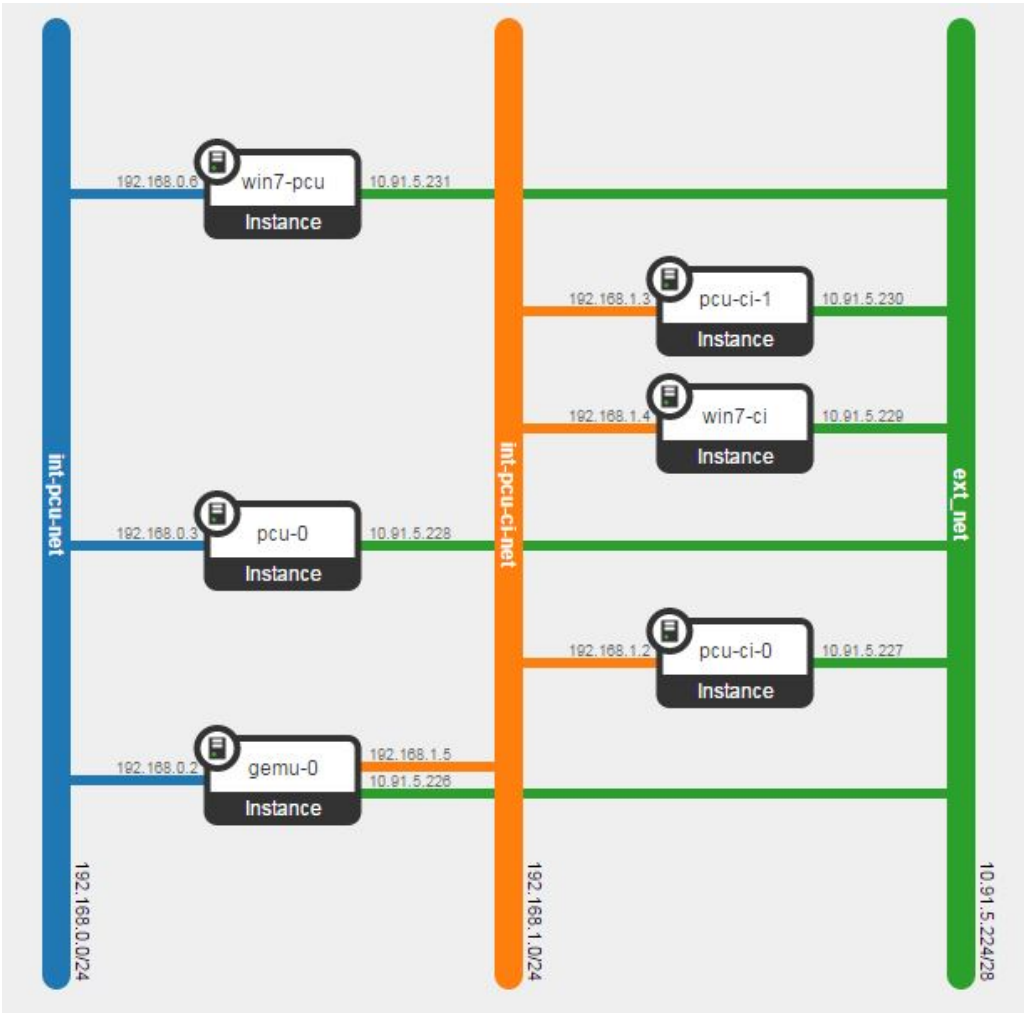

Figure 5.3 *Network Topology in OpenStack dashboard*

The manual deployment work for setting up the cloud testing environment for PCU is ready now. A post installation script can be created for possible configuration for the instance. In next subsection, we will discuss the way to deploy everything automatically at once.

5.1.2 Automated deployment

In addition to the manual way of establishing the testing environment for PCU, there is a more convenient way to reach the same goal. It is to use the OpenStack orchestration service (heat) to deploy everything needed at one time in a stack. What we need to do is to create a template with ".yaml" suffix for the file, in which the relationships and dependencies of each other should be defined. In the rest of this subsection, we will explain the general idea of automated deployment.

Based on the Heat Template Guide [4] from OpenStack official document, we know that a typical heat template should contain following structures:

- `heat_template_version`: This indicates the specified version of the template.
- `description`: This is optional to give a description of the template.
- `parameter_groups`: This is optional to specify how the input parameters should be grouped and the order of parameters.
- `parameters`: This is optional to specify input parameters that have to be provided when instantiating the template.
- `resources`: This contains the declaration of the single resources of the template.
- `outputs`: This is optional to specify output parameters available to users once the template has been instantiated.

The value of the `heat_template_version` indicates not only the format of the template but also features that will be validated and supported. Currently, there are three major versions for the heat template:

- 2013-05-23: The document contains features implemented until the Icehouse release.

- 2014-10-16: The document contains features added or/and removed until the Juno release.
- 2015-04-30: The document contains features added or/and removed until the Kilo release.

As the `parameter_groups` part is optional and can be omitted in different cases, we will directly move to the `parameters` part. This, in most cases, can provide similar functionalities as `parameter_groups`. The parameters in this part are used to customize each deployment or to bind to environment-specifics. Each parameter will be specified in a separate nested block with the name of the parameter and the additional attributes defined as nested elements. The segment below is an example of a specification for `parameters`. Except the `param name` and `type` parts, all rest of attributes can be optional in the structure. The `hidden` attribute by default will be true, but if it is defined as false, then the input parameters will be invisible for showing the stack. Users can also define restrictions for the parameter in the `constraints` part according to the requirement.

```
parameters:
  <param name>:
    type: <string | number | json | comma_delimited_list |
      boolean>
    label: <human-readable name of the parameter>
    description: <description of the parameter>
    default: <default value for parameter>
    hidden: <true | false>
    constraints:
      <parameter constraints>
```

The next part is the `resources` section. This is the mandatory part in the template to configure the actual resources, otherwise, nothing will be really done for the instantiating. The `resource ID` must be identical. The `resource type` is depending on which type of element will be deployed, for instance, that could be "OS::Nova::Server" or "OS::Neutron::Net". The `properties` part requires the resource name and its value that will be used for deploying the element, for example "image: pcu.img". The rest of attributes are optional, users can define those parts based on their own needs.

```
resources:
  <resource ID>:
    type: <resource type>
    properties:
      <property name>: <property value>
```

```
metadata:
  <resource specific metadata>
depends_on: <resource ID or list of ID>
update_policy: <update policy>
deletion_policy: <deletion policy>
```

The last section is the `outputs`, which usually could be the main parameters of the stack that gives information about what users actually created related to the stack. The syntax for defining the output is as follows.

```
outputs:
  <parameter name>:
    description: <description>
    value: <parameter value>
```

In addition to the basic template structure above, there are also intrinsic functions that can be used in the template to perform specific tasks, for example to get the value of a resource attribute at runtime. In this thesis, the general way of how it can be done will be introduced, but for more detailed information, it can be learned from the Template Guide [4].

After the template has been carefully structured, we could create the testing environment at one time by running the following command through the OpenStack heat API. The option `file` indicates the file of the template that will be used for creating this stack. Parameters are optional followed with pairs and separated by semicolon. The `stack_name` needs to be defined at last.

```
heat stack-create [-f <file>] [-P <key1=value1;key2=value2...>]
  <stack_name>
```

The result of using heat template for the deployment should not be different compared to the method of manual deployment. By this way, once we created the heat template, it can be reused in the future whenever a new set of similar deployment is needed. Besides, as all required elements have been defined in the heat template, when the stack is being created, all elements can be deployed in parallel in a few minutes. Then, a detailed structure of the stack can be found from the OpenStack dashboard orchestration part. The example in Figure 5.4 illustrates the detailed stack. All cloud elements, such as instances, networks, subnets, ports and so forth, are connected with each other according to the predefined dependencies.

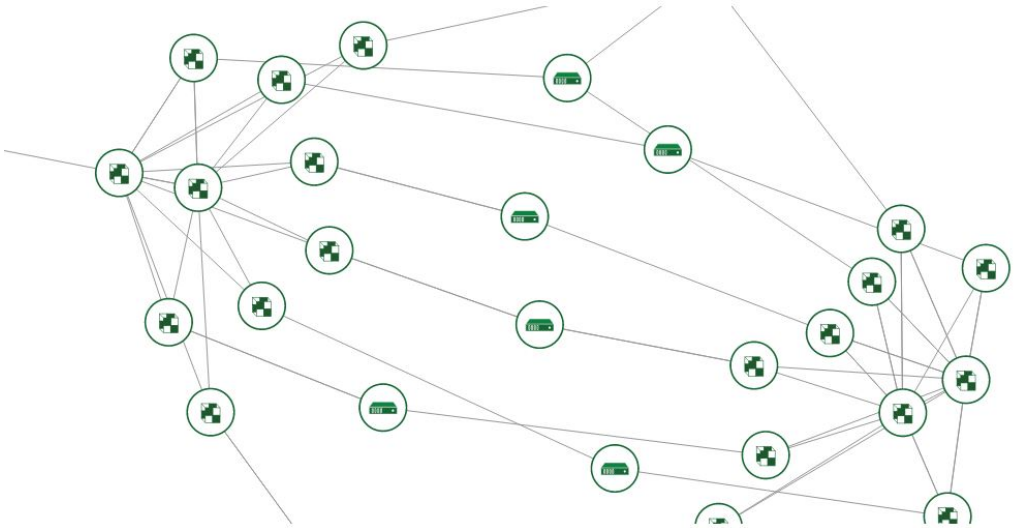

Figure 5.4 *Example of part of the heat stack*

5.2 Instances management

After the cloud based working environment has been established, the instance management looks after the operational issues, such as resilience backup and elastic scalability.

5.2.1 Resilience backup

When developing or configuring the software, for instance PCU, it is important to consider the resilience backup issue. The resilience backup is necessary when any failures or updates happen. OpenStack provides such a way to create snapshots of the currently running instance. To make the snapshot, follow the command below.

```
nova image-create <instance name or id> <name of new image>
```

After the snapshot is made for the currently running instance, it is possible to always have standby instances by deploying them from the snapshot. The way to do this is similar to launching a new instance that has been introduced previously. However previously, when launching a new instance, we will fetch the source from the image, but now we should fetch the source from the snapshot instead. The command below shows how this should be done.

```
nova boot [--flavor <flavor>] [--snapshot <snapshot-id>]
  [--nic <port-id=<port-id>>] ([--nic <port-id=<port-id>>])
  [--poll] <name>
```

To use a backup instance, extra network configuration may be needed, such as to detach the network interfaces from the original instance and attach the network interfaces to the backup instance. Thus, the original instance will be isolated as all connections to that instance will have been detached. Comparatively, if the backup instance is attached with all connections same as the previous working one, it will turn to work just like a substitute. OpenStack provides a simple way to do that by using the command below. Only the id or name of the server and the id of the port are needed to get it complete.

```
nova interface-detach <original-server> <port-id>
nova interface-attach [--port-id <port-id>] <backup-server>
```

5.2.2 Elastic scalability

Elastic scalability is a feature supported by OpenStack. One running instance can be resized in OpenStack by using the following command in the first line on-demand. Then a being resized instance will turn its status into "verify resize". Actually, a resized instance has been copied and is waiting for the confirmation. Therefore, by the second command below, the original instance will be deleted and the resized instance will be verified and active. In case of any failures, a chance to get the original instance back is possible by the third command below.

```
nova resize --poll <server> <flavor>
nova resize-confirm <server>
nova resize-revert <server>
```

However, one thing to note is that OpenStack nova scheduler is actually responsible for the instance resize. By default, it will determine the best compute node for the resized instance. In other word, when there is a single compute node, the resize for the instance will fail. However, by modifying `allow_resize_to_same_host=True` and `scheduler_default_filters=AllHostsFilter` in */etc/nova/nova.conf* file, all compute hosts will be available and unfiltered, and the same compute host is also capable of resizing as well. Nevertheless, this is recommended only for a single compute node testing environment, and not for a multi-compute environment.

6. DISCUSSION AND CONCLUSION

This thesis demonstrates that PCU can be run in the OpenStack based cloud environment in a proof-of-concept stage. All instances in the cloud have been running over months without any problems in the testing phase. We believe when it is the time to have the real cloudified PCU as a product, it will work stably there too.

With the cloudified PCU in OpenStack, software no longer needs to be designed for the dedicated hardware. The decoupling with the dedicated physical hardware enables the PCU software to be deployed in the cloud, a software defined virtualized environment. Whenever the configuration or deployment is needed for the PCU, everything can be done in the cloud, and nothing needs to be done physically. Besides, the application can only notice the logical cloud resource pool, it will never know the real hardware resources. That means all the hardware resources are centrally managed in the cloud resource pool. It is easy for the management of the resources, as well as elastic for the proper allocation of the resources. However, for the legacy product, once it has been deployed, it is fixed.

In addition, whenever there is a failure for the cloudified PCU in OpenStack, the recovery time of the new PCU will be significantly shortened. This is because that there will be always at least one standby instance which is the fully configured snapshot of the running PCU in the cloud. Thus, the backup instance can immediately take over the responsibility of the failing PCU. The time of making a new cloudified PCU in OpenStack to work will take only a few minutes, which achieves nearly "zero downtime". However, if we think about the legacy PCU, this will take hours or even days to have the new coming PCU to work. In addition to this, the CI instances are developed and tested several times per day. This guarantees continuous and fast delivery of new versions of the product. Whereas, every new release for a legacy PCU will take months to deploy.

However, there are downsides to the cloudified PCU. To achieve the completed cloudified PCU, the software architecture needs to be redesigned almost thoroughly. This will take quite a long time to accomplish. While on the other hand, it is also a good thing to be considered. As the software will be redesigned, the new version will

have more powerful functionalities, be more stable, and cater to the new technology to meet people's requirements. Another negative aspect of the cloud based PCU is that there might be always idling resources in the host server, the resources cannot be fully used. Nevertheless, with a small group of idling resources, it provides a buffer in case of extra needs. When that is needed, we do not have to worry about finding resources from elsewhere.

As OpenStack is a much powerful cloud platform with various functionalities and is evolving quite rapidly, the thesis only mentioned some of the main parts. Therefore, in future works, more services can be studied and put into use. Besides, the cloudifed product, such as PCU, can be reconstructed to be more intelligent to cater to the cloud environment. All in all, it has already been a popular trend to have the cloudified product in the telecommunication field. With cloudified products, operators can get benefits as stated above, and it will directly save their CAPEX and OPEX.

BIBLIOGRAPHY

[1] D. W. Chadwick, K. Siu, C. Lee, Y. Fouillat, and D. Germonville, "Adding federated identity management to openstack," *Grid Computing*, vol. 12, pp. 3–27, Mar. 2014.

[2] K. Chandrasekaran, *Essentials of Cloud Computing*. CRC Press, 2014.

[3] F. Chong, G. Carraro, and R. Wolter, "Multi-tenant data architecture," MSDN Architecture Center, Jun. 2006, [Online] Accessed on: 24.4.2015. Available at: https://msdn.microsoft.com/en-us/library/aa479086.aspx.

[4] Template guide. [Online] Accessed on: 8.5.2015. Available at: http://docs.openstack.org/developer/heat/template_guide/index.html.

[5] K. Jackson and C. Bunch, *OpenStack Cloud Computing Cookbook*, 2nd ed. Packt Publishing, Oct. 2013.

[6] A. Johansson. (2014, Dec.) Selinux. CentOS. [Online] Accessed on: 19.3.2015. Available at: http://wiki.centos.org/HowTos.

[7] R. Krebs, C. Momm, and S. Kounev, "Architectural concerns in multi-tenant saas applications," in *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, 2012, pp. 426–431.

[8] K.-I. Ku, W.-H. Choi, M. Chung, K. Kim, W.-Y. Kim, and S.-J. Hur, "Method for distribution, execution and management ofthe customized application based on software virtualization," in *13th International Conference on Advanced Communication Technology*. IEEE, Feb. 2010, pp. 493–496.

[9] V. Lahtinen, "Specifying packet switched user-plane over packet abis interface for bsc," Master's thesis, Tampere University of Technology, 2010.

[10] G. Lee, *Cloud Networking: Understanding Cloud-based Data Center Networks*. Elsevier Inc., 2014, ch. 1,2.

[11] Neutron-linux-bridge-plugin. [Online] Accessed on: 2.4.2015. Available at: https://wiki.openstack.org/wiki/Neutron-Linux-Bridge-Plugin.

[12] S. Mathys, "Packstack," Presentation, OpenStack, Apr. 2013, [Online] Accessed on: 22.3.2015. Available at: https://wiki.openstack.org/wiki/Packstack.

[13] P. Mell and T. Grance, "The nist definition of cloud computing," Special Publication 800-145, National Institue of Standards and Technology (NIST), Sep. 2011.

[14] Neutron/ml2. [Online] Accessed on: 2.4.2015. Available at: https://wiki.openstack.org/wiki/Neutron/ML2.

[15] What is open vswitch? [Online] Accessed on: 2.4.2015. Available at: http://www.openvswitch.org/.

[16] Companies supporting the openstack foundation. [Online] Accessed on: 25.3.2015. Available at: http://www.openstack.org/foundation/companies/.

[17] Openstack dashboard. [Online] Accessed on: 13.4.2015. Available at: https://www.openstack.org/software/openstack-dashboard/.

[18] *OpenStack Cloud Administrator Guide*, Manual, OpenStack Foundation, Mar. 2015, accessed on: 30.3.2015. Available at: http://docs.openstack.org/admin-guide-cloud/content/index.html.

[19] *OpenStack Command-Line Interface Reference*, Manual, OpenStack Foundation, Apr. 2015, accessed on: 30.4.2015. Available at: http://docs.openstack.org/cli-reference/content/index.html.

[20] *OpenStack Installation Guide for Red Hat Enterprise Linux 7, CentOS 7, and Fedora 20 - Juno*, Manual, OpenStack Foundation, Mar. 2015, accessed on: 19.3.2015. Available at: http://docs.openstack.org/juno/install-guide/install/yum/content/ch_basic_environment.html.

[21] *OpenStack Virtual Machine Image Guide*, Manual, OpenStack Foundation, Apr. 2015, accessed on: 1.4.2015. Available at: http://docs.openstack.org/image-guide/content/index.html.

[22] Openstack: The open source cloud operating system. [Online] Accessed on: 25.3.2015. Available at: http://www.openstack.org/software/.

[23] Release cycle. [Online] Accessed on: 25.3.2015. Available at: https://wiki.openstack.org/wiki/Release_Cycle.

[24] Openstack roadmap. [Online] Accessed on: 22.5.2015. Available at: http://www.openstack.org/software/roadmap/.

[25] Openstack storage. [Online] Accessed on: 2.4.2015. Available at: https://www.openstack.org/software/openstack-storage/.

[26] J. Penttinen, *GPRS in Wireless Data: Functioning and Design of the Network*, 1st ed. WSOY, 2002.

[27] Adding a compute node. Red Hat, Inc. [Online] Accessed on: 22.3.2015. Available at: https://www.rdoproject.org/Adding_a_compute_node.

[28] Neutron with existing external network. Red Hat, Inc. [Online] Accessed on: 23.3.2015. Available at: https://www.rdoproject.org/Neutron_with_existing_external_network.

[29] Rdo quickstart. Red Hat, Inc. [Online] Accessed on: 19.3.2015. Available at: https://www.rdoproject.org/Quickstart.

[30] Red hat + centos. Red Hat, Inc. [Online] Accessed on: 19.3.2015. Available at: https://community.redhat.com/centos-faq/.

[31] A. Semnanian, J. Pham, B. Englert, and X. Wu, "Virtualization technology and its impact on computer hardware architecture," in *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on.* IEEE, Apr. 2011, pp. 719–724.

[32] E. Seurre, P. Savelli, and P.-J. Pietri, *GPRS for Mobile Internet.* Artech House Print on Demand, Dec. 2002.

[33] SMG, "Digital cellular telecommunications system (phase 2+); general packet radio service (gprs) requirements specification of gprs (gsm 01.60 version 6.0.0)," European Telecommunications Standards Institute (ETSI), Tech. Rep. TR 101 186, Apr. 1998.

[34] "Understanding full virtualization, paravirtualization, and hardware assist," White paper, VMware, Sep. 2007.

[35] "Vmware esx and vmware esxi," Product Datasheet, VMware, 2009.

[36] "Vmware thinapp agentless application virtualization overview," White paper, VMware, 2010.

[37] D. Yu, J. Wang, B. Hu, J. Lit, X. Zhang, K. He, and L.-J. Zhang, "A practical architecture of cloudification of legacy applications," in *Services (SERVICES), 2011 IEEE World Congress on.* IEEE, Jul. 2011, pp. 17–24.

APPENDIX A. ANSWER FILE

[general]

Path to a Public key to install on servers. If a usable key has not been installed on the remote servers the user will be prompted for a password and this key will be installed so the password will not be required again
CONFIG_SSH_KEY=

Set a default password everywhere. The default password will be overriden by whatever password is set for each individual service or user.
CONFIG_DEFAULT_PASSWORD=

# Set to 'y' if you would like Packstack to install MariaDB
CONFIG_MARIADB_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Image Service (Glance)
CONFIG_GLANCE_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Block Storage (Cinder)
CONFIG_CINDER_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Compute (Nova)
CONFIG_NOVA_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Networking (Neutron). Otherwise Nova Network will be used.
CONFIG_NEUTRON_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Dashboard (Horizon)
CONFIG_HORIZON_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Object Storage (Swift)
CONFIG_SWIFT_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Metering (Ceilometer)
CONFIG_CEILOMETER_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Orchestration (Heat)
CONFIG_HEAT_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Clustering (Sahara)
CONFIG_SAHARA_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Database (Trove)
CONFIG_TROVE_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Bare Metal (Ironic)
CONFIG_IRONIC_INSTALL=n

# Set to 'y' if you would like Packstack to install the OpenStack Client packages. An admin "rc" file will also be installed
CONFIG_CLIENT_INSTALL=y

# Comma separated list of NTP servers. Leave plain if Packstack should not install ntpd on instances.
CONFIG_NTP_SERVERS=10.8.147.40

# Set to 'y' if you would like Packstack to install Nagios to monitor OpenStack hosts
CONFIG_NAGIOS_INSTALL=y

Comma separated list of servers to be excluded from installation in case you are running Packstack the second time with the same answer file and don't want Packstack to touch these servers. Leave plain if you don't need to exclude any server.
EXCLUDE_SERVERS=

# Set to 'y' if you want to run OpenStack services in debug mode. Otherwise set to 'n'.
CONFIG_DEBUG_MODE=n

# The IP address of the server on which to install OpenStack services specific to controller role such as API servers, Horizon, etc.
CONFIG_CONTROLLER_HOST=10.91.5.218

# The list of IP addresses of the server on which to install the Nova compute service
CONFIG_COMPUTE_HOSTS=10.91.5.218,10.91.5.222

# The list of IP addresses of the server on which to install the network service such as Nova network or Neutron
CONFIG_NETWORK_HOSTS=10.91.5.218

Set to 'y' if you want to use VMware vCenter as hypervisor and storage. Otherwise

set to 'n'.
CONFIG_VMWARE_BACKEND=n

# Set to 'y' if you want to use unsupported parameters. This should be used only if you know what you are doing.Issues caused by using unsupported options won't be fixed before next major release.
CONFIG_UNSUPPORTED=n

The IP address of the VMware vCenter server
CONFIG_VCENTER_HOST=

The username to authenticate to VMware vCenter server
CONFIG_VCENTER_USER=

The password to authenticate to VMware vCenter server
CONFIG_VCENTER_PASSWORD=

The name of the vCenter cluster
CONFIG_VCENTER_CLUSTER_NAME=

# (Unsupported!) The IP address of the server on which to install OpenStack services specific to storage servers such as Glance and Cinder.
CONFIG_STORAGE_HOST=10.91.5.218

# (Unsupported!) The IP address of the server on which to install OpenStack services specific to Sahara
CONFIG_SAHARA_HOST=10.91.5.218

# To subscribe each server to EPEL enter "y"
CONFIG_USE_EPEL=y

A comma separated list of URLs to any additional yum repositories to install
CONFIG_REPO=

To subscribe each server with Red Hat subscription manager, include this with CONFIG_RH_PW
CONFIG_RH_USER=

To subscribe each server with RHN Satellite,fill Satellite's URL here. Note that either satellite's username/password or activation key has to be provided
CONFIG_SATELLITE_URL=

To subscribe each server with Red Hat subscription manager, include this with

```
CONFIG_RH_USER
CONFIG_RH_PW=

# To enable RHEL optional repos use value "y"
CONFIG_RH_OPTIONAL=y

# Specify a HTTP proxy to use with Red Hat subscription manager
CONFIG_RH_PROXY=

# Specify port of Red Hat subscription manager HTTP proxy
CONFIG_RH_PROXY_PORT=

# Specify a username to use with Red Hat subscription manager HTTP proxy
CONFIG_RH_PROXY_USER=

# Specify a password to use with Red Hat subscription manager HTTP proxy
CONFIG_RH_PROXY_PW=

# Username to access RHN Satellite
CONFIG_SATELLITE_USER=

# Password to access RHN Satellite
CONFIG_SATELLITE_PW=

# Activation key for subscription to RHN Satellite
CONFIG_SATELLITE_AKEY=

# Specify a path or URL to a SSL CA certificate to use
CONFIG_SATELLITE_CACERT=

# If required specify the profile name that should be used as an identifier for the
system in RHN Satellite
CONFIG_SATELLITE_PROFILE=

# Comma separated list of flags passed to rhnreg_ks. Valid flags are: novirtinfo,
norhnsd, nopackages
CONFIG_SATELLITE_FLAGS=

# Specify a HTTP proxy to use with RHN Satellite
CONFIG_SATELLITE_PROXY=

# Specify a username to use with an authenticated HTTP proxy
CONFIG_SATELLITE_PROXY_USER=
```

```
# Specify a password to use with an authenticated HTTP proxy.
CONFIG_SATELLITE_PROXY_PW=

# Set the AMQP service backend. Allowed values are: qpid, rabbitmq
CONFIG_AMQP_BACKEND=rabbitmq

# The IP address of the server on which to install the AMQP service
CONFIG_AMQP_HOST=10.91.5.218

# Enable SSL for the AMQP service
CONFIG_AMQP_ENABLE_SSL=n

# Enable Authentication for the AMQP service
CONFIG_AMQP_ENABLE_AUTH=n

# The password for the NSS certificate database of the AMQP service
CONFIG_AMQP_NSS_CERTDB_PW=PW_PLACEHOLDER

# The port in which the AMQP service listens to SSL connections
CONFIG_AMQP_SSL_PORT=5671

# The filename of the certificate that the AMQP service is going to use
CONFIG_AMQP_SSL_CERT_FILE=/etc/pki/tls/certs/amqp_selfcert.pem

# The filename of the private key that the AMQP service is going to use
CONFIG_AMQP_SSL_KEY_FILE=/etc/pki/tls/private/amqp_selfkey.pem

# Auto Generates self signed SSL certificate and key
CONFIG_AMQP_SSL_SELF_SIGNED=y

# User for amqp authentication
CONFIG_AMQP_AUTH_USER=amqp_user

# Password for user authentication
CONFIG_AMQP_AUTH_PASSWORD=PW_PLACEHOLDER

# The IP address of the server on which to install MariaDB or IP address of DB
server to use if MariaDB installation was not selected
CONFIG_MARIADB_HOST=10.91.5.218

# Username for the MariaDB admin user
CONFIG_MARIADB_USER=root
```

```
# Password for the MariaDB admin user
CONFIG_MARIADB_PW=******

# The password to use for the Keystone to access DB
CONFIG_KEYSTONE_DB_PW=******

# Region name
CONFIG_KEYSTONE_REGION=RegionOne

# The token to use for the Keystone service api
CONFIG_KEYSTONE_ADMIN_TOKEN=******

# The password to use for the Keystone admin user
CONFIG_KEYSTONE_ADMIN_PW=******

# The password to use for the Keystone demo user
CONFIG_KEYSTONE_DEMO_PW=******

# Kestone token format. Use either UUID or PKI
CONFIG_KEYSTONE_TOKEN_FORMAT=UUID

# Name of service to use to run keystone (keystone or httpd)
CONFIG_KEYSTONE_SERVICE_NAME=keystone

# The password to use for the Glance to access DB
CONFIG_GLANCE_DB_PW=******

# The password to use for the Glance to authenticate with Keystone
CONFIG_GLANCE_KS_PW=******

# Glance storage backend controls how Glance stores disk images. Supported values:
file, swift. Note that Swift installation have to be enabled to have swift backend
working. Otherwise Packstack will fallback to 'file'.
CONFIG_GLANCE_BACKEND=file

# The password to use for the Cinder to access DB
CONFIG_CINDER_DB_PW=******

# The password to use for the Cinder to authenticate with Keystone
CONFIG_CINDER_KS_PW=******

# The Cinder backend to use, valid options are: lvm, gluster, nfs, netapp
CONFIG_CINDER_BACKEND=lvm
```

# Create Cinder's volumes group. This should only be done for testing on a proof-of-concept installation of Cinder. This will create a file-backed volume group and is not suitable for production usage.
CONFIG_CINDER_VOLUMES_CREATE=y

# Cinder's volumes group size. Note that actual volume size will be extended with 3
CONFIG_CINDER_VOLUMES_SIZE=20G

A single or comma separated list of gluster volume shares to mount, eg: ip-address:/vol-name, domain:/vol-name
CONFIG_CINDER_GLUSTER_MOUNTS=

A single or comma seprated list of NFS exports to mount, eg: ip-address:/export-name CONFIG_CINDER_NFS_MOUNTS=

(required) Administrative user account name used to access the storage system or proxy server.
CONFIG_CINDER_NETAPP_LOGIN=

(required) Password for the administrative user account specified in the netapp_login parameter.
CONFIG_CINDER_NETAPP_PASSWORD=

(required) The hostname (or IP address) for the storage system or proxy server.
CONFIG_CINDER_NETAPP_HOSTNAME=

# (optional) The TCP port to use for communication with ONTAPI on the storage system. Traditionally, port 80 is used for HTTP and port 443 is used for HTTPS; however, this value should be changed if an alternate port has been configured on the storage system or proxy server. Defaults to 80.
CONFIG_CINDER_NETAPP_SERVER_PORT=80

# (optional) The storage family type used on the storage system; valid values are ontap_7mode for using Data ONTAP operating in 7-Mode or ontap_cluster for using clustered Data ONTAP, or eseries for NetApp E-Series. Defaults to ontap_cluster.
CONFIG_CINDER_NETAPP_STORAGE_FAMILY=ontap_cluster

# (optional) The transport protocol used when communicating with ONTAPI on the storage system or proxy server. Valid values are http or https. Defaults to http.
CONFIG_CINDER_NETAPP_TRANSPORT_TYPE=http

# (optional) The storage protocol to be used on the data path with the storage system; valid values are iscsi or nfs. Defaults to nfs.
CONFIG_CINDER_NETAPP_STORAGE_PROTOCOL=nfs

# (optional) The quantity to be multiplied by the requested volume size to ensure enough space is available on the virtual storage server (Vserver) to fulfill the volume creation request. Defaults to 1.0.
CONFIG_CINDER_NETAPP_SIZE_MULTIPLIER=1.0

# (optional) This parameter specifies the threshold for last access time for images in the NFS image cache. When a cache cleaning cycle begins, images in the cache that have not been accessed in the last M minutes, where M is the value of this parameter, will be deleted from the cache to create free space on the NFS share. Defaults to 720.
CONFIG_CINDER_NETAPP_EXPIRY_THRES_MINUTES=720

# (optional) If the percentage of available space for an NFS share has dropped below the value specified by this parameter, the NFS image cache will be cleaned. Defaults to 20
CONFIG_CINDER_NETAPP_THRES_AVL_SIZE_PERC_START=20

# (optional) When the percentage of available space on an NFS share has reached the percentage specified by this parameter, the driver will stop clearing files from the NFS image cache that have not been accessed in the last M minutes, where M is the value of the expiry_thres_minutes parameter. Defaults to 60.
CONFIG_CINDER_NETAPP_THRES_AVL_SIZE_PERC_STOP=60

(optional) File with the list of available NFS shares. Defaults to ".
CONFIG_CINDER_NETAPP_NFS_SHARES_CONFIG=

(optional) This parameter is only utilized when the storage protocol is configured to use iSCSI. This parameter is used to restrict provisioning to the specified controller volumes. Specify the value of this parameter to be a comma separated list of NetApp controller volume names to be used for provisioning. Defaults to ".
CONFIG_CINDER_NETAPP_VOLUME_LIST=

(optional) The vFiler unit on which provisioning of block storage volumes will be done. This parameter is only used by the driver when connecting to an instance with a storage family of Data ONTAP operating in 7-Mode and the storage protocol selected is iSCSI. Only use this parameter when utilizing the MultiStore feature on the NetApp storage system. Defaults to ".

CONFIG_CINDER_NETAPP_VFILER=

(optional) This parameter specifies the virtual storage server (Vserver) name on the storage cluster on which provisioning of block storage volumes should occur. If using the NFS storage protocol, this parameter is mandatory for storage service catalog support (utilized by Cinder volume type extra_specs support). If this parameter is specified, the exports belonging to the Vserver will only be used for provisioning in the future. Block storage volumes on exports not belonging to the Vserver specified by this parameter will continue to function normally. Defaults to ''.
CONFIG_CINDER_NETAPP_VSERVER=

(optional) This option is only utilized when the storage family is configured to eseries. This option is used to restrict provisioning to the specified controllers. Specify the value of this option to be a comma separated list of controller hostnames or IP addresses to be used for provisioning. Defaults to ''.
CONFIG_CINDER_NETAPP_CONTROLLER_IPS=

(optional) Password for the NetApp E-Series storage array. Defaults to ''.
CONFIG_CINDER_NETAPP_SA_PASSWORD=

# (optional) This option is used to specify the path to the E-Series proxy application on a proxy server. The value is combined with the value of the netapp_transport_type, netapp_server_hostname, and netapp_server_port options to create the URL used by the driver to connect to the proxy application. Defaults to '/devmgr/v2'.
CONFIG_CINDER_NETAPP_WEBSERVICE_PATH=/devmgr/v2

(optional) This option is used to restrict provisioning to the specified storage pools. Only dynamic disk pools are currently supported. Specify the value of this option to be a comma separated list of disk pool names to be used for provisioning. Defaults to ''.
CONFIG_CINDER_NETAPP_STORAGE_POOLS=

CONFIG_IRONIC_DB_PW=PW_PLACEHOLDER

# The password to use for Ironic to authenticate with Keystone
CONFIG_IRONIC_KS_PW=PW_PLACEHOLDER

# The password to use for the Nova to access DB
CONFIG_NOVA_DB_PW=******

# The password to use for the Nova to authenticate with Keystone
CONFIG_NOVA_KS_PW=******

# The overcommitment ratio for virtual to physical CPUs. Set to 1.0 to disable CPU overcommitment
CONFIG_NOVA_SCHED_CPU_ALLOC_RATIO=16.0

# The overcommitment ratio for virtual to physical RAM. Set to 1.0 to disable RAM overcommitment
CONFIG_NOVA_SCHED_RAM_ALLOC_RATIO=1.5

# Protocol used for instance migration. Allowed values are tcp and ssh. Note that by defaul nova user is created with /sbin/nologin shell so that ssh protocol won't be working. To make ssh protocol work you have to fix nova user on compute hosts manually.
CONFIG_NOVA_COMPUTE_MIGRATE_PROTOCOL=tcp

# The manager that will run nova compute.
CONFIG_NOVA_COMPUTE_MANAGER=nova.compute.manager.Compute-Manager

# Private interface for Flat DHCP on the Nova compute servers
CONFIG_NOVA_COMPUTE_PRIVIF=enp3s0f1

# Nova network manager
CONFIG_NOVA_NETWORK_MANAGER=nova.network.manager.FlatDHCP-Manager

# Public interface on the Nova network server
CONFIG_NOVA_NETWORK_PUBIF=enp3s0f0

# Private interface for network manager on the Nova network server
CONFIG_NOVA_NETWORK_PRIVIF=enp3s0f1

# IP Range for network manager
CONFIG_NOVA_NETWORK_FIXEDRANGE=192.168.32.0/22

# IP Range for Floating IP's
CONFIG_NOVA_NETWORK_FLOATRANGE=10.3.4.0/22

# Automatically assign a floating IP to new instances
CONFIG_NOVA_NETWORK_AUTOASSIGNFLOATINGIP=n

```
# First VLAN for private networks
CONFIG_NOVA_NETWORK_VLAN_START=100

# Number of networks to support
CONFIG_NOVA_NETWORK_NUMBER=1

# Number of addresses in each private subnet
CONFIG_NOVA_NETWORK_SIZE=255

# The password to use for Neutron to authenticate with Keystone
CONFIG_NEUTRON_KS_PW=******

# The password to use for Neutron to access DB
CONFIG_NEUTRON_DB_PW=******

# The name of the ovs bridge (or empty for linuxbridge) that the Neutron L3 agent
will use for external traffic, or 'provider' using provider networks.
CONFIG_NEUTRON_L3_EXT_BRIDGE=br-ex

# Neutron metadata agent password
CONFIG_NEUTRON_METADATA_PW=******

# Set to 'y' if you would like Packstack to install Neutron LBaaS
CONFIG_LBAAS_INSTALL=n

# Set to 'y' if you would like Packstack to install Neutron L3 Metering agent
CONFIG_NEUTRON_METERING_AGENT_INSTALL=y

# Whether to configure neutron Firewall as a Service
CONFIG_NEUTRON_FWAAS=n

# A comma separated list of network type driver entrypoints to be loaded from the
neutron.ml2.type_drivers namespace.
CONFIG_NEUTRON_ML2_TYPE_DRIVERS=flat,vlan,vxlan

# A comma separated ordered list of network_types to allocate as tenant networks.
The value 'local' is only useful for single-box testing but provides no connectivity
between hosts.
CONFIG_NEUTRON_ML2_TENANT_NETWORK_TYPES=vxlan

# A comma separated ordered list of networking mechanism driver entrypoints to
be loaded from the neutron.ml2.mechanism_drivers namespace.
CONFIG_NEUTRON_ML2_MECHANISM_DRIVERS=openvswitch
```

# A comma separated list of physical_network names with which flat networks can be created. Use * to allow flat networks with arbitrary physical_network names.
CONFIG_NEUTRON_ML2_FLAT_NETWORKS=*

# A comma separated list of <physical_network>:<vlan_min>:<vlan_max> or <physical_network> specifying physical_network names usable for VLAN provider and tenant networks, as well as ranges of VLAN tags on each available for allocation to tenant networks.
CONFIG_NEUTRON_ML2_VLAN_RANGES=physnet1:2020:2030

A comma separated list of <tun_min>:<tun_max> tuples enumerating ranges of GRE tunnel IDs that are available for tenant network allocation. Should be an array with tun_max +1 - tun_min > 1000000
CONFIG_NEUTRON_ML2_TUNNEL_ID_RANGES=

Multicast group for VXLAN. If unset, disables VXLAN enable sending allocate broadcast traffic to this multicast group. When left unconfigured, will disable multicast VXLAN mode. Should be an Multicast IP (v4 or v6) address.
CONFIG_NEUTRON_ML2_VXLAN_GROUP=

# A comma separated list of <vni_min>:<vni_max> tuples enumerating ranges of VXLAN VNI IDs that are available for tenant network allocation. Min value is 0 and Max value is 16777215.
CONFIG_NEUTRON_ML2_VNI_RANGES=10:100

# The name of the L2 agent to be used with Neutron
CONFIG_NEUTRON_L2_AGENT=openvswitch

A comma separated list of interface mappings for the Neutron linuxbridge plugin (eg. physnet1:eth1,physnet2:eth2,physnet3:eth3)
CONFIG_NEUTRON_LB_INTERFACE_MAPPINGS=

# A comma separated list of bridge mappings for the Neutron openvswitch plugin (eg. physnet1:br-eth1,physnet2:br-eth2,physnet3 :br-eth3)
CONFIG_NEUTRON_OVS_BRIDGE_MAPPINGS=physnet1:br-ex

# A comma separated list of colon-separated OVS bridge:interface pairs. The interface will be added to the associated bridge.
CONFIG_NEUTRON_OVS_BRIDGE_IFACES=br-ex:enp3s0f1

The interface for the OVS tunnel. Packstack will override the IP address used for tunnels on this hypervisor to the IP found on the specified interface. (eg. eth1)

CONFIG_NEUTRON_OVS_TUNNEL_IF=enp3s0f0

# VXLAN UDP port
CONFIG_NEUTRON_OVS_VXLAN_UDP_PORT=4789

# To set up Horizon communication over https set this to 'y'
CONFIG_HORIZON_SSL=n

PEM encoded certificate to be used for ssl on the https server, leave blank if one should be generated, this certificate should not require a passphrase
CONFIG_SSL_CERT=

SSL keyfile corresponding to the certificate if one was entered
CONFIG_SSL_KEY=

PEM encoded CA certificates from which the certificate chain of the server certificate can be assembled.
CONFIG_SSL_CACHAIN=

# The password to use for the Swift to authenticate with Keystone
CONFIG_SWIFT_KS_PW=******

A comma separated list of devices which to use as Swift Storage device. Each entry should take the format /path/to/dev, for example /dev/vdb will install /dev/vdb as Swift storage device (packstack does not create the filesystem, you must do this first). If value is omitted Packstack will create a loopback device for test setup
CONFIG_SWIFT_STORAGES=

# Number of swift storage zones, this number MUST be no bigger than the number of storage devices configured
CONFIG_SWIFT_STORAGE_ZONES=1

# Number of swift storage replicas, this number MUST be no bigger than the number of storage zones configured
CONFIG_SWIFT_STORAGE_REPLICAS=1

# FileSystem type for storage nodes
CONFIG_SWIFT_STORAGE_FSTYPE=ext4

# Shared secret for Swift
CONFIG_SWIFT_HASH=******

Size of the swift loopback file storage device

CONFIG_SWIFT_STORAGE_SIZE=2G

# The password used by Heat user to authenticate against MySQL
CONFIG_HEAT_DB_PW=PW_PLACEHOLDER

# The encryption key to use for authentication info in database
CONFIG_HEAT_AUTH_ENC_KEY=******

# The password to use for the Heat to authenticate with Keystone
CONFIG_HEAT_KS_PW=PW_PLACEHOLDER

# Set to 'y' if you would like Packstack to install Heat CloudWatch API
CONFIG_HEAT_CLOUDWATCH_INSTALL=n

# Set to 'y' if you would like Packstack to install Heat CloudFormation API
CONFIG_HEAT_CFN_INSTALL=y

# Name of Keystone domain for Heat
CONFIG_HEAT_DOMAIN=heat

# Name of Keystone domain admin user for Heat
CONFIG_HEAT_DOMAIN_ADMIN=heat_admin

# Password for Keystone domain admin user for Heat
CONFIG_HEAT_DOMAIN_PASSWORD=PW_PLACEHOLDER

# Whether to provision for demo usage and testing. Note that provisioning is only supported for all-in-one installations.
CONFIG_PROVISION_DEMO=n

# Whether to configure tempest for testing
CONFIG_PROVISION_TEMPEST=n

The name of the Tempest Provisioning user. If you don't provide a user name, Tempest will be configured in a standalone mode
CONFIG_PROVISION_TEMPEST_USER=

# The password to use for the Tempest Provisioning user
CONFIG_PROVISION_TEMPEST_USER_PW=******

# The CIDR network address for the floating IP subnet
CONFIG_PROVISION_DEMO_FLOATRANGE=172.24.4.224/28

```
# A URL or local file location for the Cirros demo image used for Glance
CONFIG_PROVISION_CIRROS_URL=http://download.cirros-cloud.net/0.3.3/-
cirros-0.3.3-x86_64-disk.img

# The uri of the tempest git repository to use
CONFIG_PROVISION_TEMPEST_REPO_URI=https://github.com/openstack-
/tempest.git

# The revision of the tempest git repository to use
CONFIG_PROVISION_TEMPEST_REPO_REVISION=master

# Whether to configure the ovs external bridge in an all-in-one deployment
CONFIG_PROVISION_ALL_IN_ONE_OVS_BRIDGE=n

# The password to use for the Sahara DB access
CONFIG_SAHARA_DB_PW=PW_PLACEHOLDER

# The password to use for Sahara to authenticate with Keystone
CONFIG_SAHARA_KS_PW=PW_PLACEHOLDER

# Secret key for signing metering messages
CONFIG_CEILOMETER_SECRET=******

# The password to use for Ceilometer to authenticate with Keystone
CONFIG_CEILOMETER_KS_PW=******

# Backend driver for group membership coordination
CONFIG_CEILOMETER_COORDINATION_BACKEND=redis

# The IP address of the server on which to install MongoDB
CONFIG_MONGODB_HOST=10.91.5.218

# The IP address of the server on which to install redis
CONFIG_REDIS_HOST=10.91.5.218

# The port on which the redis server listens
CONFIG_REDIS_PORT=6379

# The password to use for the Trove DB access
CONFIG_TROVE_DB_PW=PW_PLACEHOLDER

# The password to use for Trove to authenticate with Keystone
CONFIG_TROVE_KS_PW=PW_PLACEHOLDER
```

```
# The user to use when Trove connects to Nova
CONFIG_TROVE_NOVA_USER=admin

# The tenant to use when Trove connects to Nova
CONFIG_TROVE_NOVA_TENANT=services

# The password to use when Trove connects to Nova
CONFIG_TROVE_NOVA_PW=PW_PLACEHOLDER

# The password of the nagiosadmin user on the Nagios server
CONFIG_NAGIOS_PW=******
```